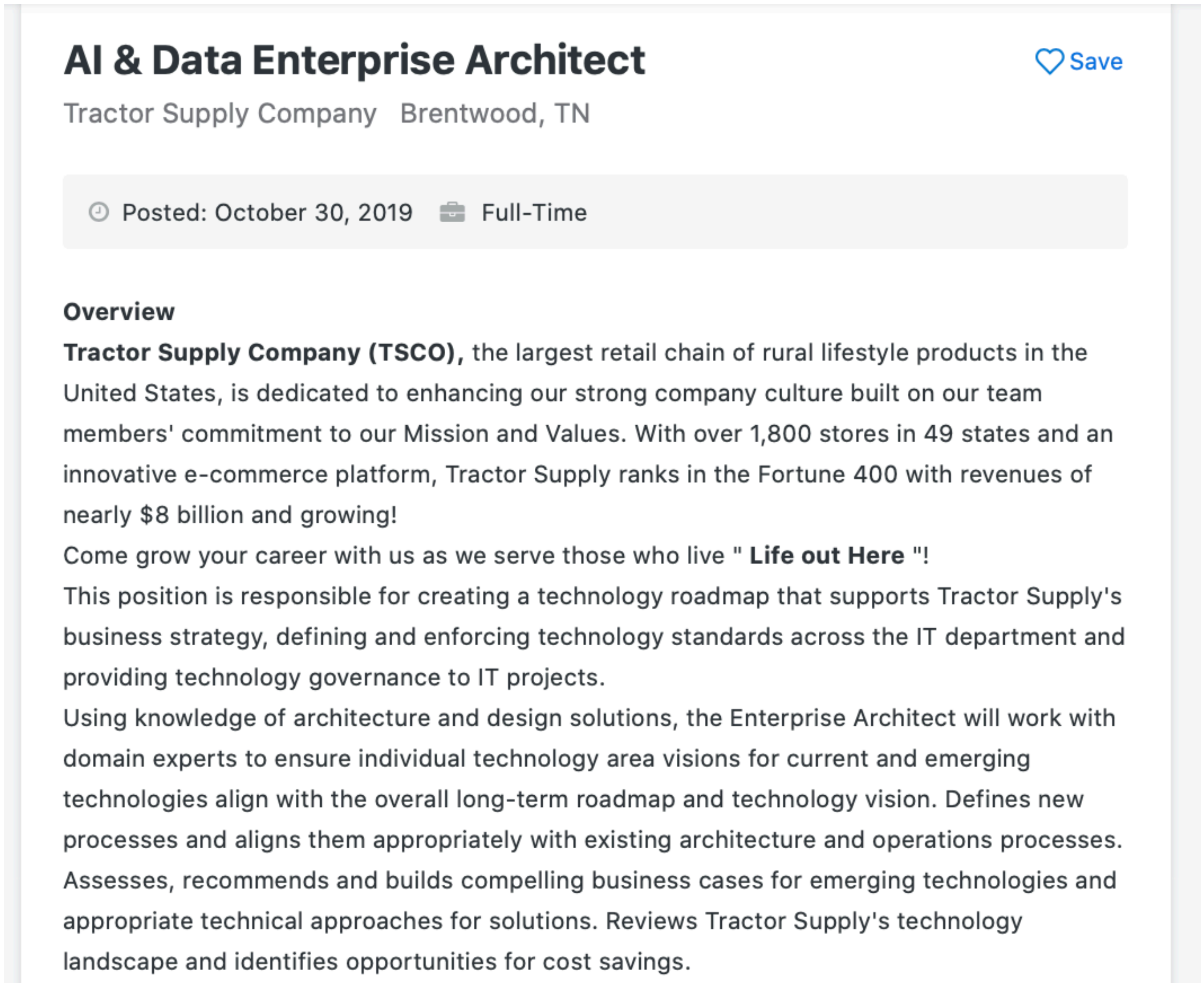
EXHIBIT B

US8089980	Tractor Supply Company ("Accused Party")
1. A method for protection switching of geographically separate switching systems arranged in pairs, comprising:	On information and belief, the accused party utilizes a system that practices a method for protection switching of geographically separate switching systems (e.g., distributed or remote racks for Datanodes) arranged in pairs (e.g., racks are arranged in pairs). On information and belief, the accused party utilizes Hadoop HDFS.

AI & Data Enterprise Architect

Save

Tractor Supply Company Brentwood, TN

Posted: October 30, 2019 Full-Time

Overview

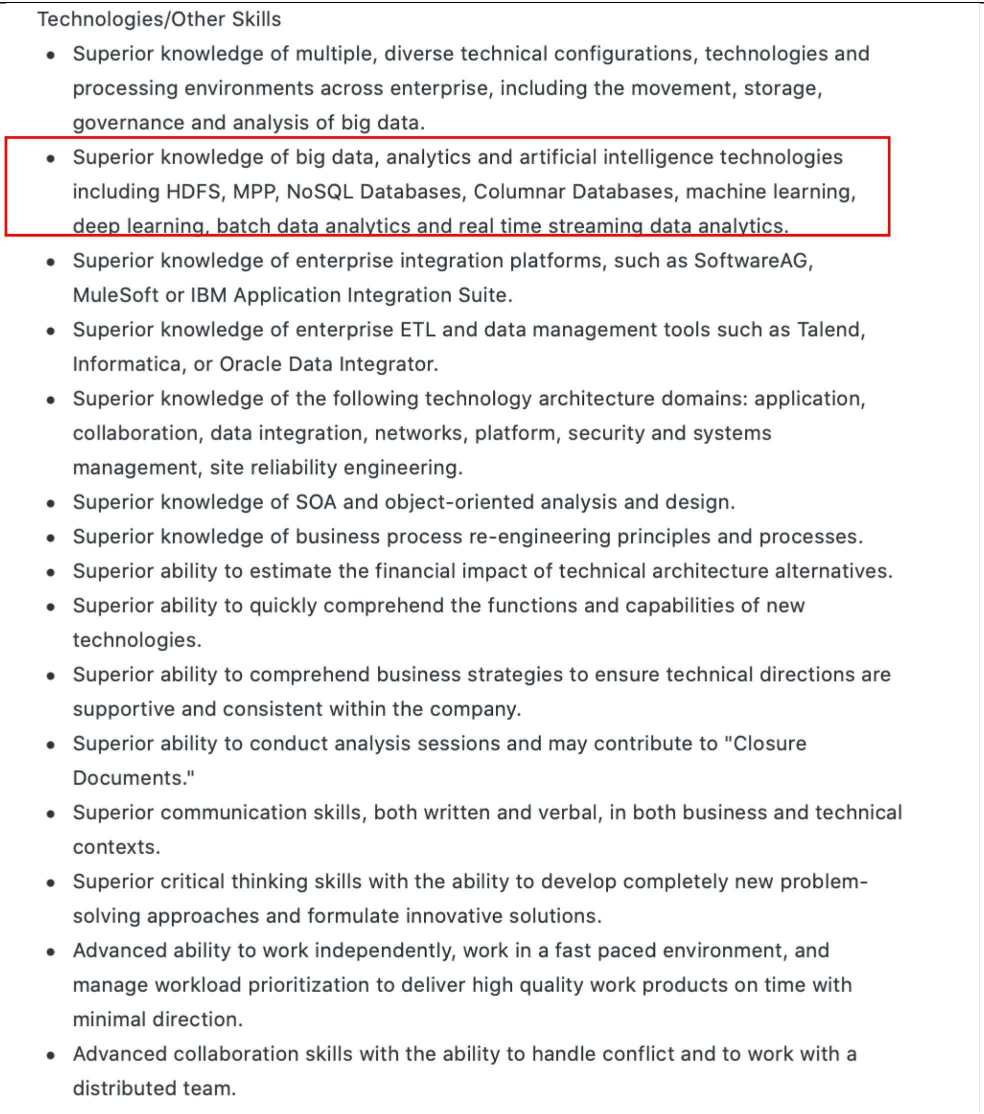
Tractor Supply Company (TSCO), the largest retail chain of rural lifestyle products in the United States, is dedicated to enhancing our strong company culture built on our team members' commitment to our Mission and Values. With over 1,800 stores in 49 states and an innovative e-commerce platform, Tractor Supply ranks in the Fortune 400 with revenues of nearly $8 billion and growing!

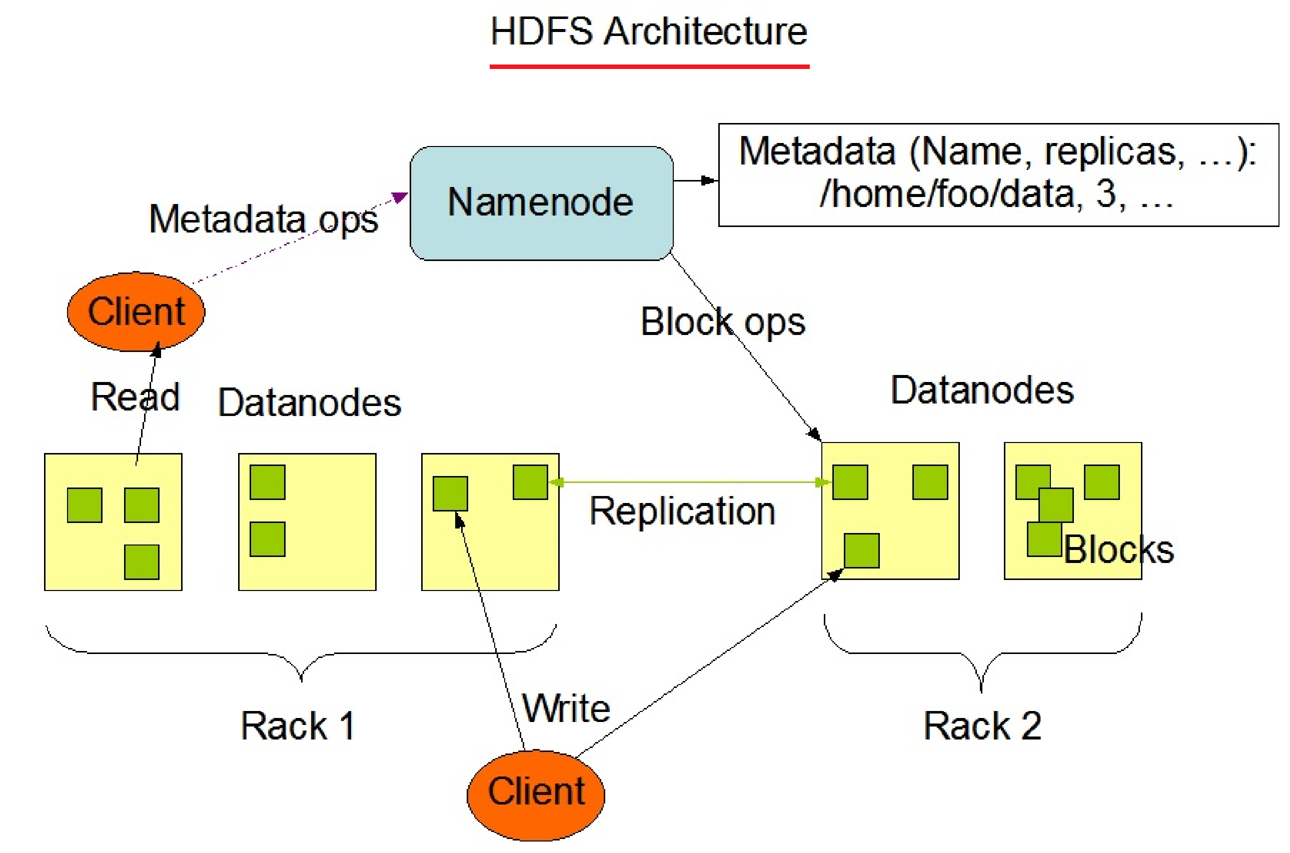
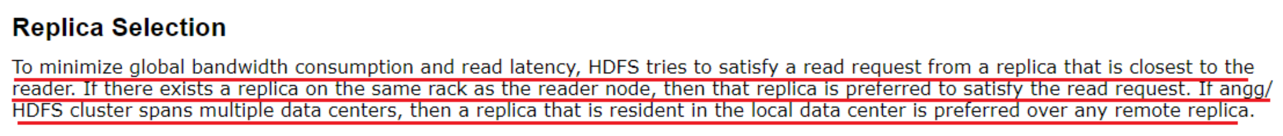
Come grow your career with us as we serve those who live " **Life out Here** "!

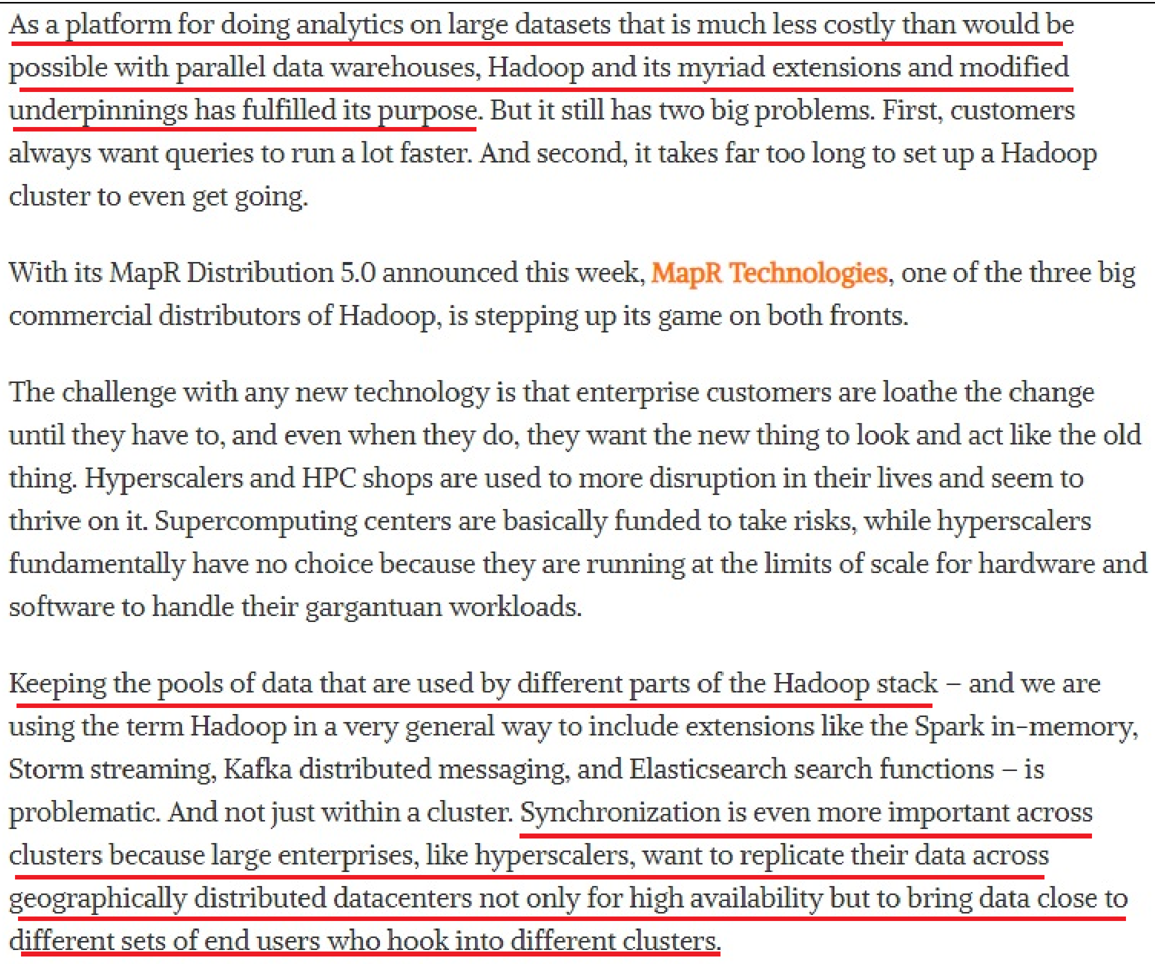
This position is responsible for creating a technology roadmap that supports Tractor Supply's business strategy, defining and enforcing technology standards across the IT department and providing technology governance to IT projects.

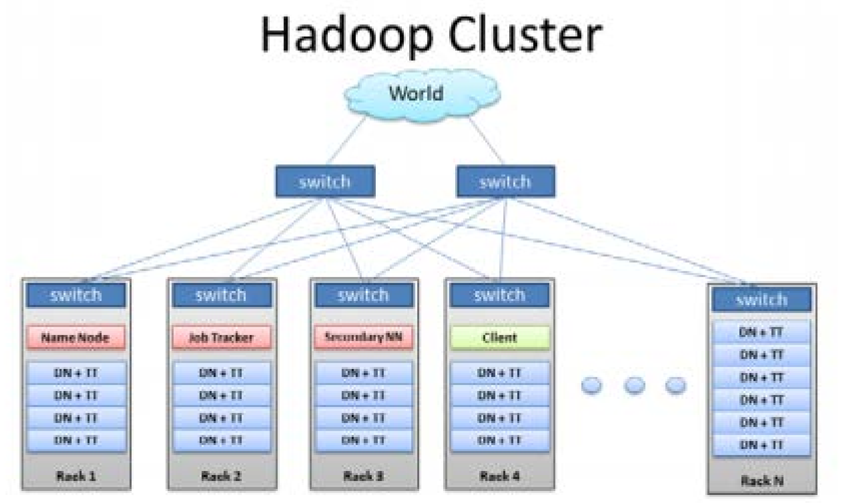
Using knowledge of architecture and design solutions, the Enterprise Architect will work with domain experts to ensure individual technology area visions for current and emerging technologies align with the overall long-term roadmap and technology vision. Defines new processes and aligns them appropriately with existing architecture and operations processes. Assesses, recommends and builds compelling business cases for emerging technologies and appropriate technical approaches for solutions. Reviews Tractor Supply's technology landscape and identifies opportunities for cost savings.

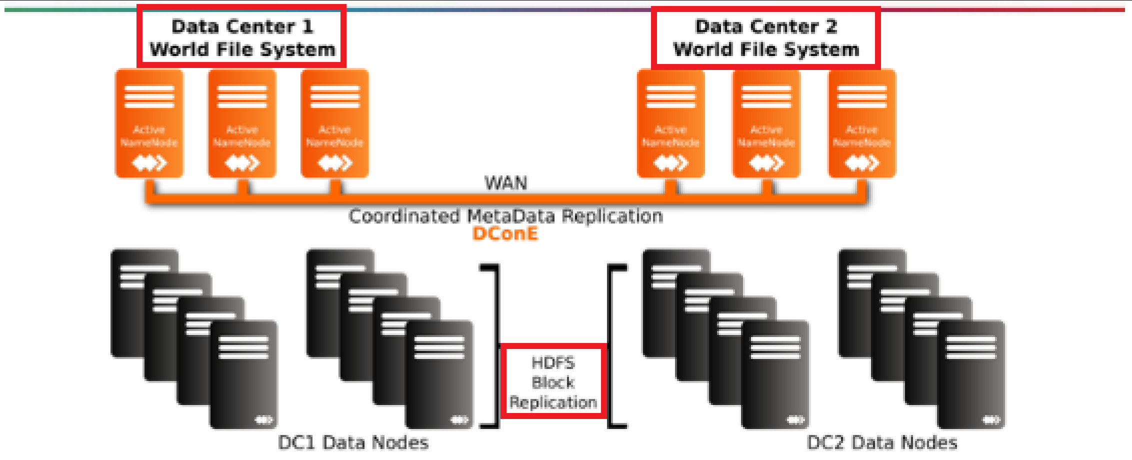
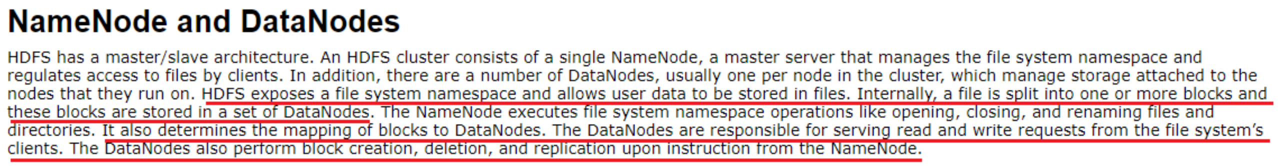
https://www.ziprecruiter.com/c/Tractor-Supply-Company/Job/AI-&-Data-Enterprise-Architect/-in-Brentwood,TN?jid=DO35c088eb547b06ee3fc106e7d9d95cb6

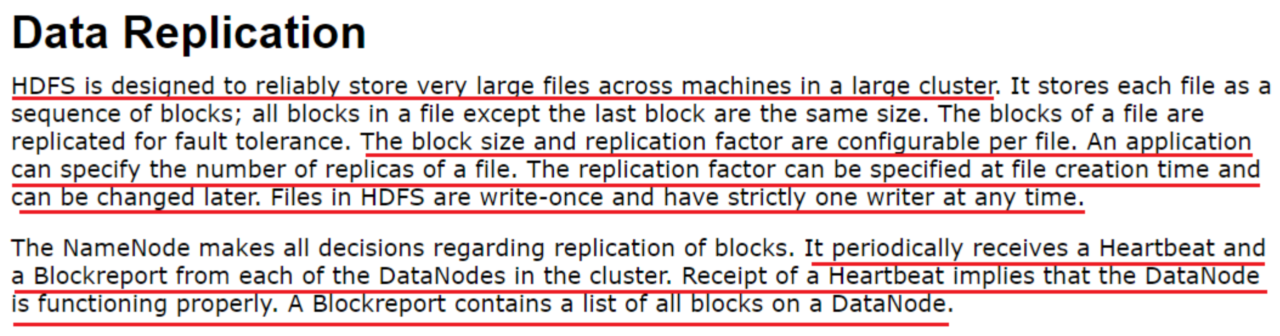
	 Technologies/Other Skills • Superior knowledge of multiple, diverse technical configurations, technologies and processing environments across enterprise, including the movement, storage, governance and analysis of big data. • Superior knowledge of big data, analytics and artificial intelligence technologies including HDFS, MPP, NoSQL Databases, Columnar Databases, machine learning, deep learning, batch data analytics and real time streaming data analytics. • Superior knowledge of enterprise integration platforms, such as SoftwareAG, MuleSoft or IBM Application Integration Suite. • Superior knowledge of enterprise ETL and data management tools such as Talend, Informatica, or Oracle Data Integrator. • Superior knowledge of the following technology architecture domains: application, collaboration, data integration, networks, platform, security and systems management, site reliability engineering. • Superior knowledge of SOA and object-oriented analysis and design. • Superior knowledge of business process re-engineering principles and processes. • Superior ability to estimate the financial impact of technical architecture alternatives. • Superior ability to quickly comprehend the functions and capabilities of new technologies. • Superior ability to comprehend business strategies to ensure technical directions are supportive and consistent within the company. • Superior ability to conduct analysis sessions and may contribute to "Closure Documents." • Superior communication skills, both written and verbal, in both business and technical contexts. • Superior critical thinking skills with the ability to develop completely new problem-solving approaches and formulate innovative solutions. • Advanced ability to work independently, work in a fast paced environment, and manage workload prioritization to deliver high quality work products on time with minimal direction. • Advanced collaboration skills with the ability to handle conflict and to work with a distributed team. https://www.ziprecruiter.com/c/Tractor-Supply-Company/Job/AI-&-Data-Enterprise-Architect/-in-Brentwood,TN?jid=DO35c088eb547b06ee3fc106e7d9d95cb6

	 HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, …): /home/foo/data, 3, … Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html **Replica Selection** To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request. If angg/ HDFS cluster spans multiple data centers, then a replica that is resident in the local data center is preferred over any remote replica. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

	 As a platform for doing analytics on large datasets that is much less costly than would be possible with parallel data warehouses, Hadoop and its myriad extensions and modified underpinnings has fulfilled its purpose. But it still has two big problems. First, customers always want queries to run a lot faster. And second, it takes far too long to set up a Hadoop cluster to even get going. With its MapR Distribution 5.0 announced this week, MapR Technologies, one of the three big commercial distributors of Hadoop, is stepping up its game on both fronts. The challenge with any new technology is that enterprise customers are loathe the change until they have to, and even when they do, they want the new thing to look and act like the old thing. Hyperscalers and HPC shops are used to more disruption in their lives and seem to thrive on it. Supercomputing centers are basically funded to take risks, while hyperscalers fundamentally have no choice because they are running at the limits of scale for hardware and software to handle their gargantuan workloads. Keeping the pools of data that are used by different parts of the Hadoop stack – and we are using the term Hadoop in a very general way to include extensions like the Spark in-memory, Storm streaming, Kafka distributed messaging, and Elasticsearch search functions – is problematic. And not just within a cluster. Synchronization is even more important across clusters because large enterprises, like hyperscalers, want to replicate their data across geographically distributed datacenters not only for high availability but to bring data close to different sets of end users who hook into different clusters. https://www.nextplatform.com/2015/06/09/synchronizing-data-in-hadoop-lakes/

	 Hadoop Cluster World switch switch switch Name Node DN + TT Rack 1 Job Tracker Rack 2 Secondary NN Rack 3 Client Rack 4 Rack N https://its.northeastern.edu/researchcomputing/wp-content/uploads/2018/07/HDFS_Discovery_Cluster_Nilay_Roy.pdf

	 Data Center 1 World File System Data Center 2 World File System Active NameNode WAN Coordinated MetaData Replication DConE HDFS Block Replication DC1 Data Nodes DC2 Data Nodes **Step 1** Metadata modifications coordinated using a cross data center quorum of Non-Stop NameNode **Step 2** Client creates file, adds block, write pipeline is established in local data center DataNodes, data is written and local write pipeline closes **Step 3** Asynchronous block replicator pushes one block to foreign data center https://slideplayer.com/slide/4336686/ **NameNode and DataNodes** HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

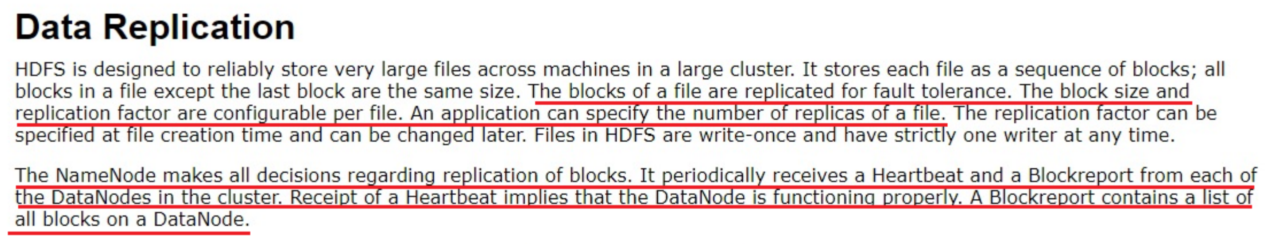


Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

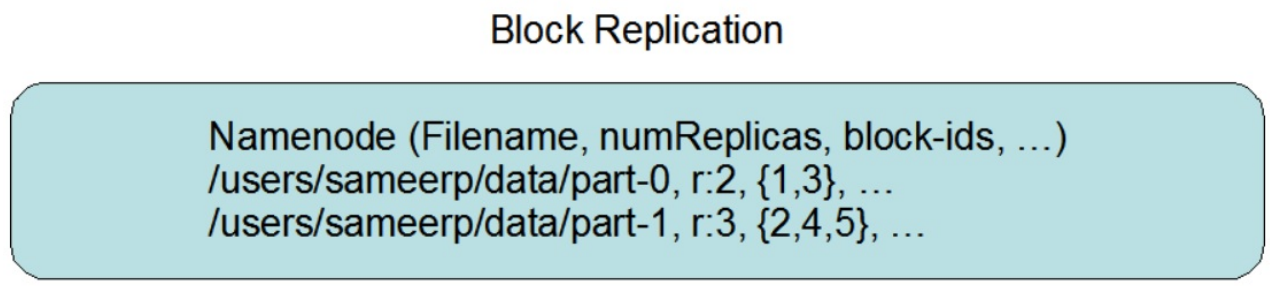
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html



Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

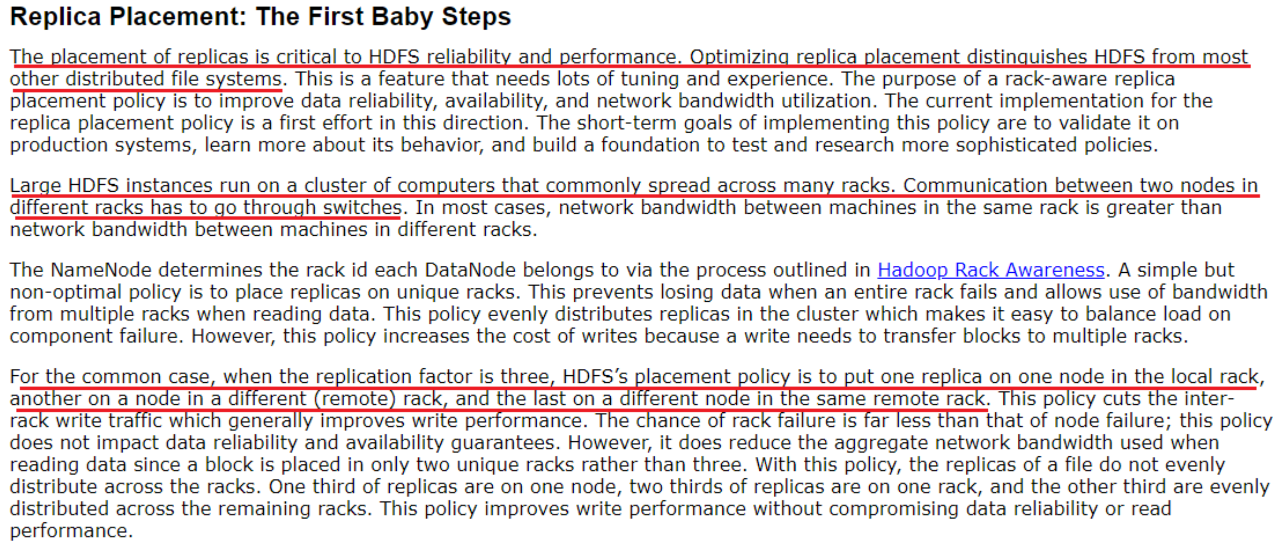
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.



Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



Replica Placement: The First Baby Steps

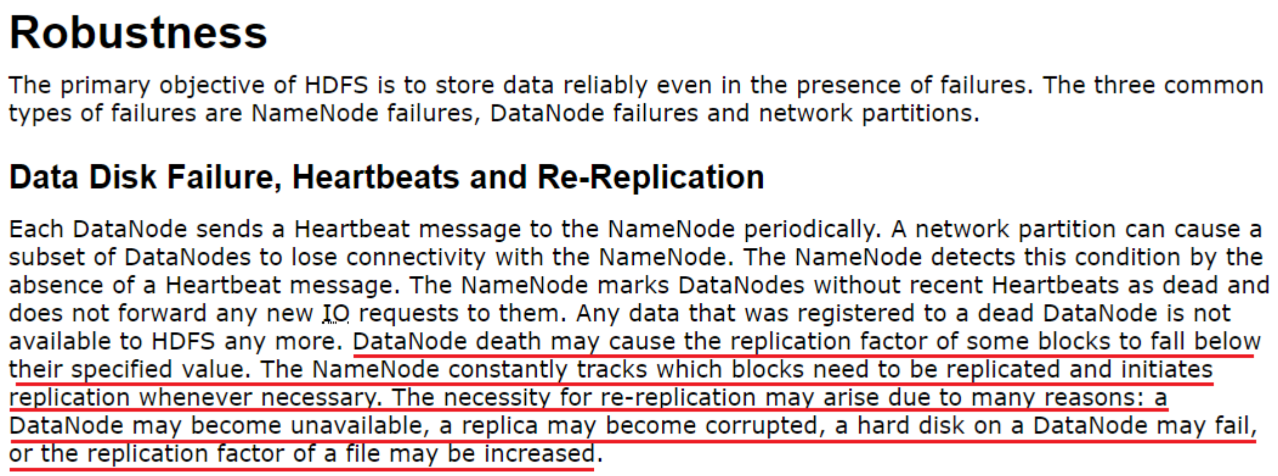
The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



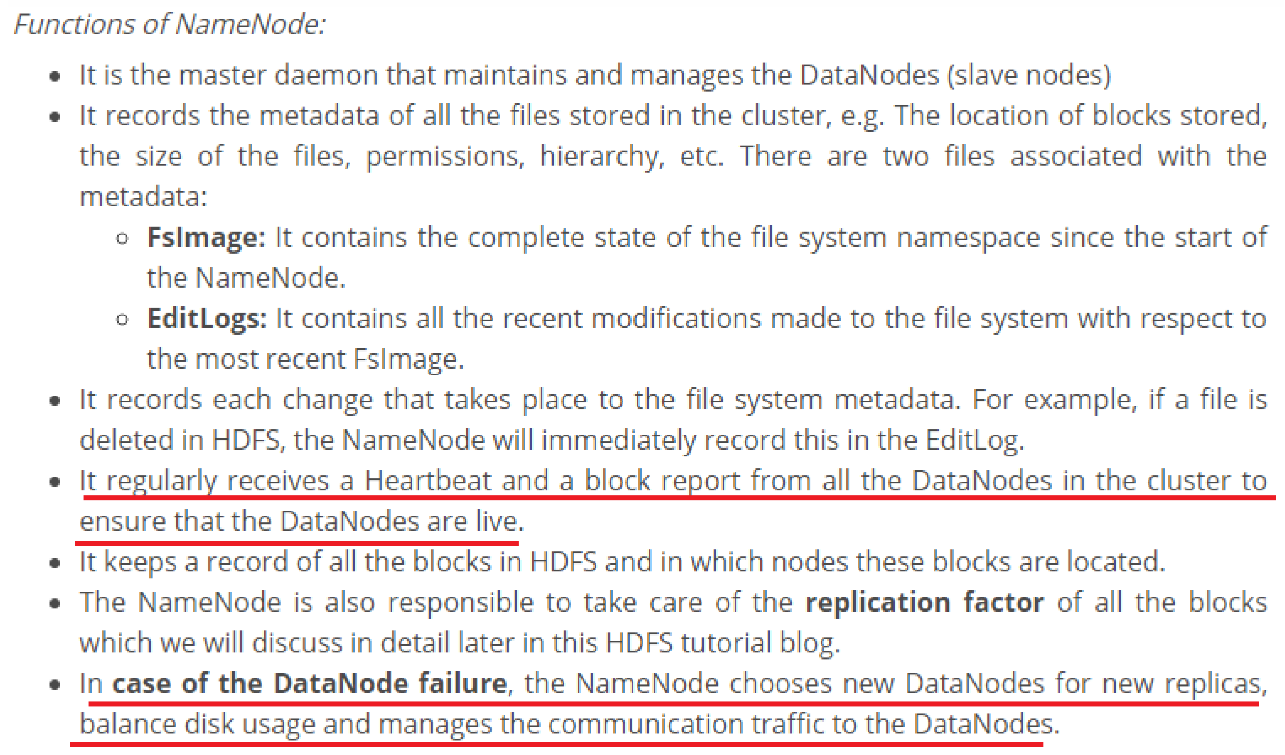
Robustness

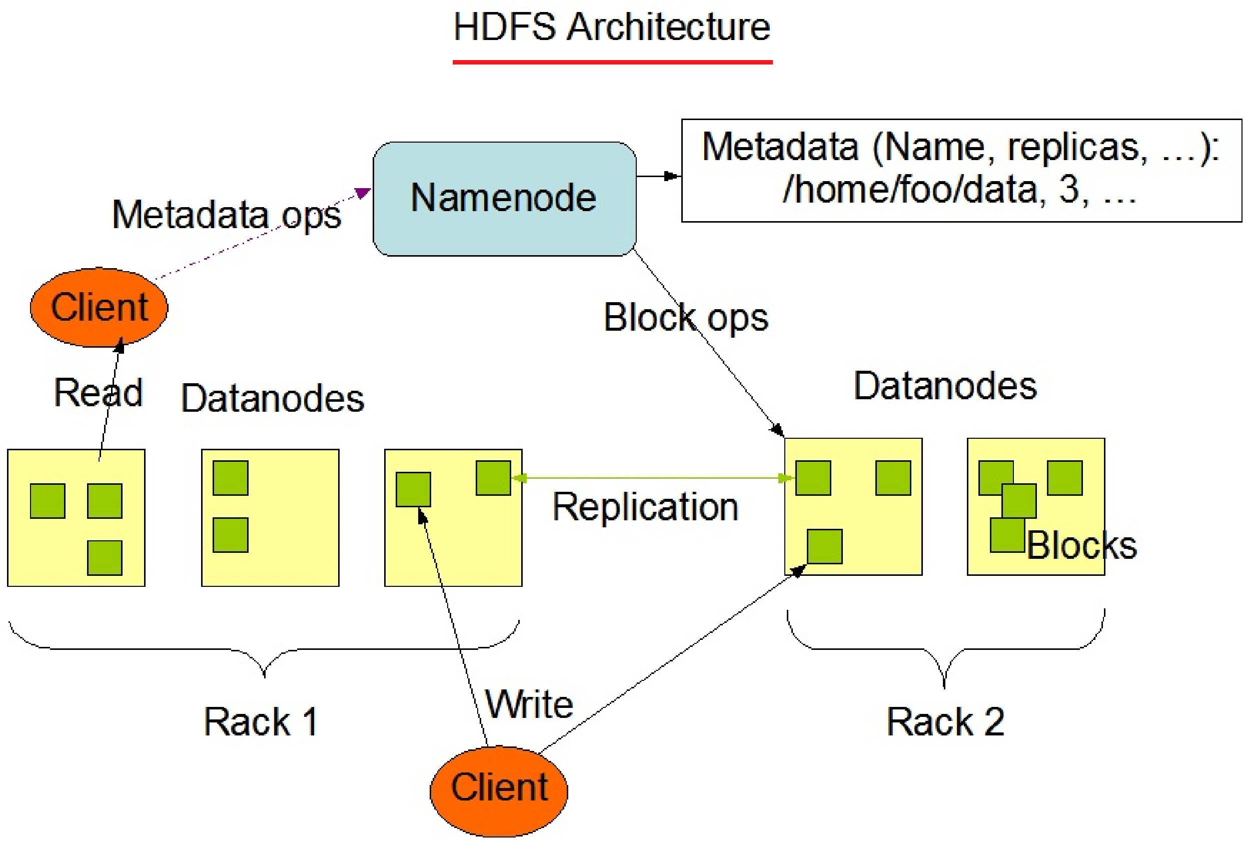
The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

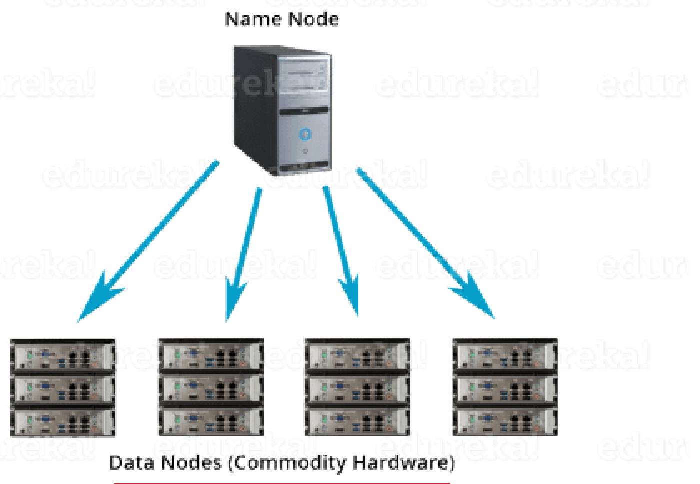
Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Safemode

	 *Functions of NameNode:* • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata: ◦ **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode. ◦ **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/
providing a pair of switching systems which are geographically separate and which supply a dedicated redundancy to each other, one of the pair of switching systems is in an active operating state and the other is	The system, at least in internal testing and usage, utilized by the accused product practices providing a pair of switching systems (e.g., racks for Datanodes are arranged in pair) which are geographically separate (e.g., distributed or remote racks for Datanodes) and which supply a dedicated redundancy to each other, one of the pair of switching systems is in an active operating state (e.g., a local rack for data node) and the other is in a hot-standby operating state (e.g., a remote rack for data node). As shown below, Hadoop distributed file system (HDFS) architecture provides data replication at Data nodes for failure protection. A replication factor represents number of replicas of a file at different Data nodes. The replication factor is 3 for a file by default. A first replica is stored at a Data

in a hot-standby operating state;	node in a local rack (e.g., active operating state) and two replicas at two different Data nodes in a remote rack (e.g., hot-standby state). The two racks for data nodes are distributed or remote to each other. The data nodes in remote rack keep their state synchronized with the data node in local rack to perform fast failover.  HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, …): /home/foo/data, 3, … Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

HDFS Architecture:



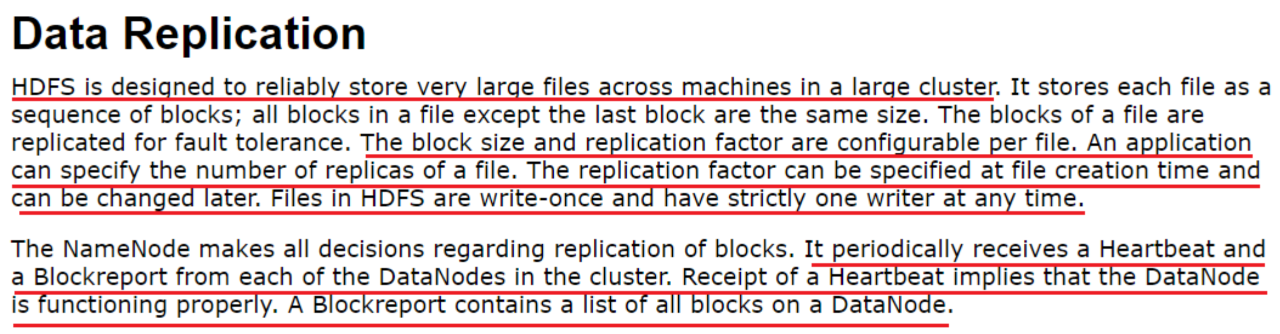
Apache HDFS or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

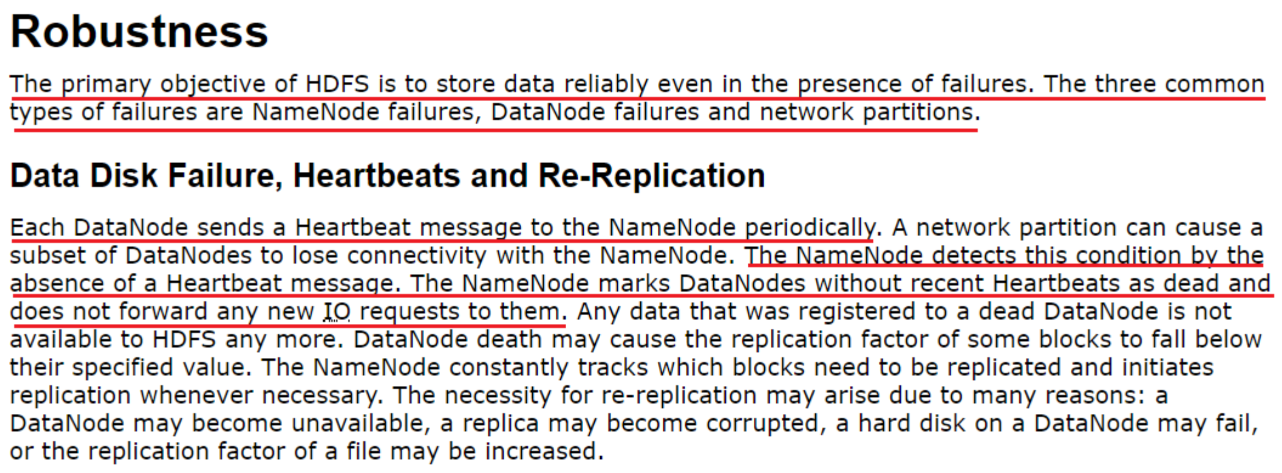


Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html



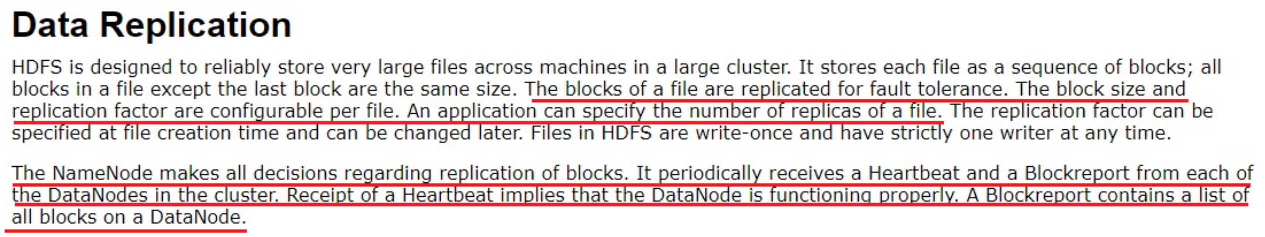
Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

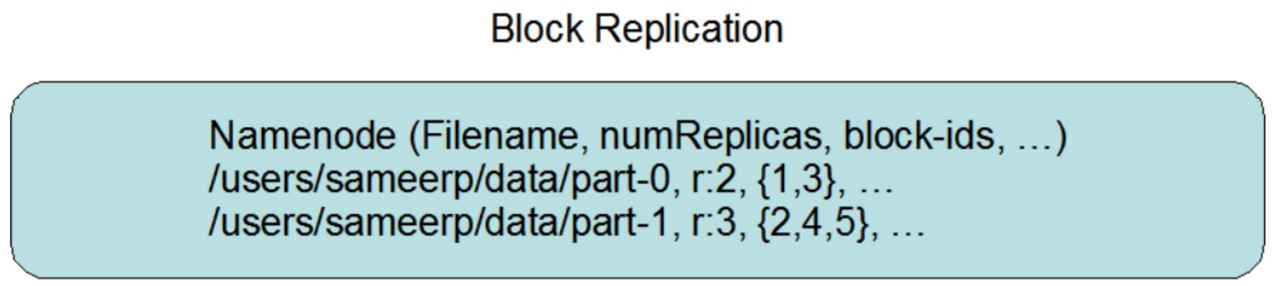
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Disk+Failure%2C+Heartbeats+and+Re-Replication



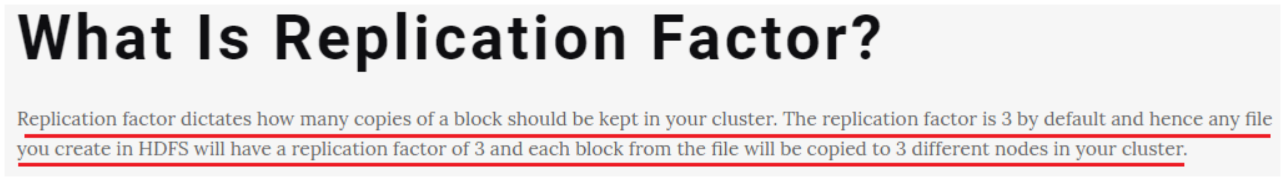
Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

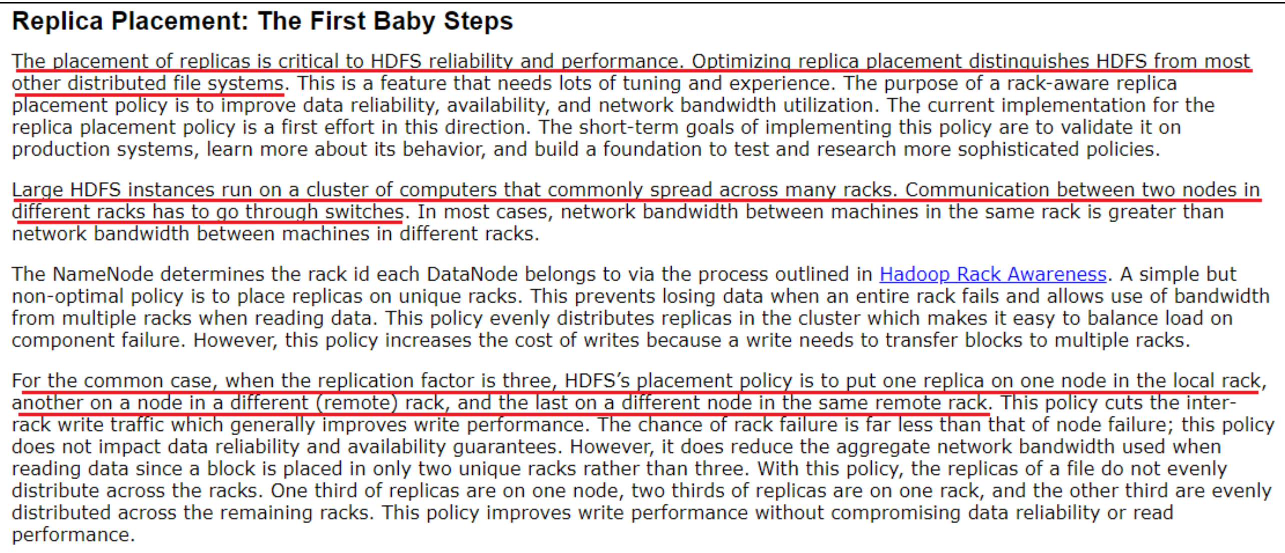
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



What Is Replication Factor?

Replication factor dictates how many copies of a block should be kept in your cluster. The replication factor is 3 by default and hence any file you create in HDFS will have a replication factor of 3 and each block from the file will be copied to 3 different nodes in your cluster.

https://www.hadoopinrealworld.com/how-to-change-default-replication-factor/



Replica Placement: The First Baby Steps

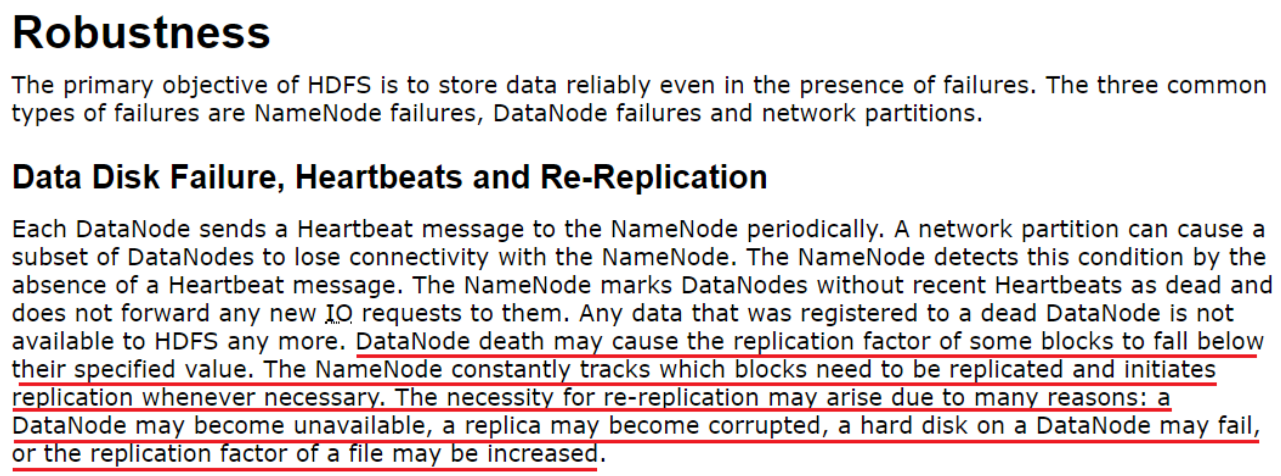
The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



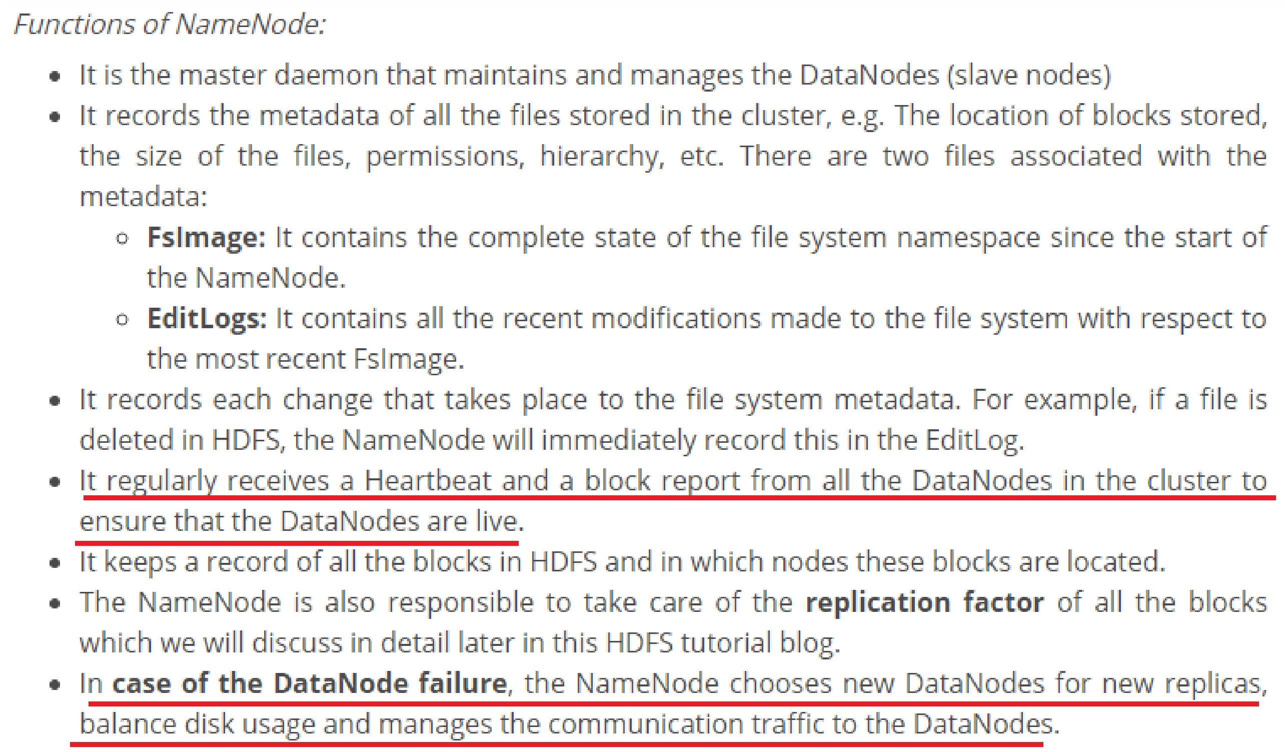
Robustness

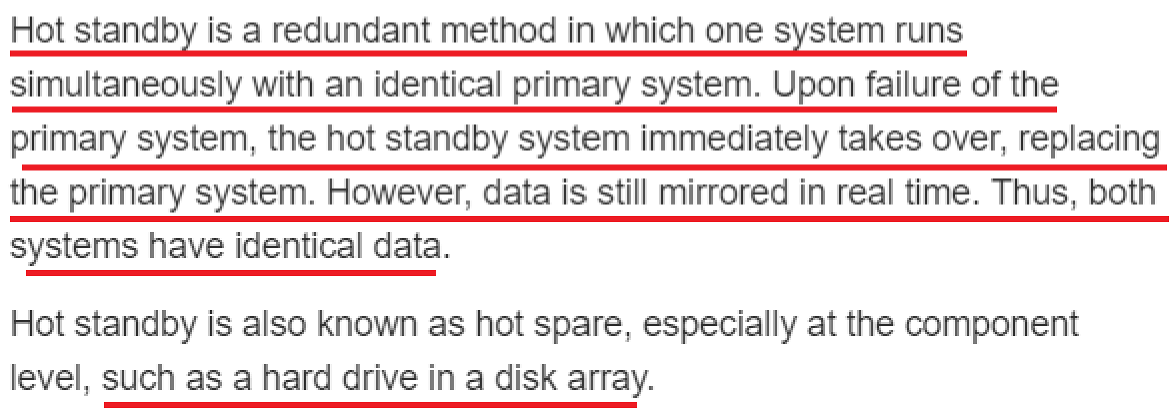
The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

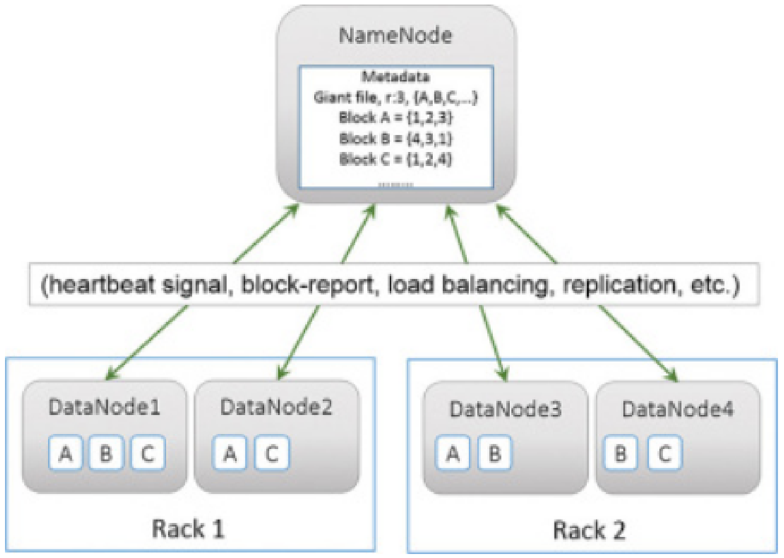
Data Disk Failure, Heartbeats and Re-Replication

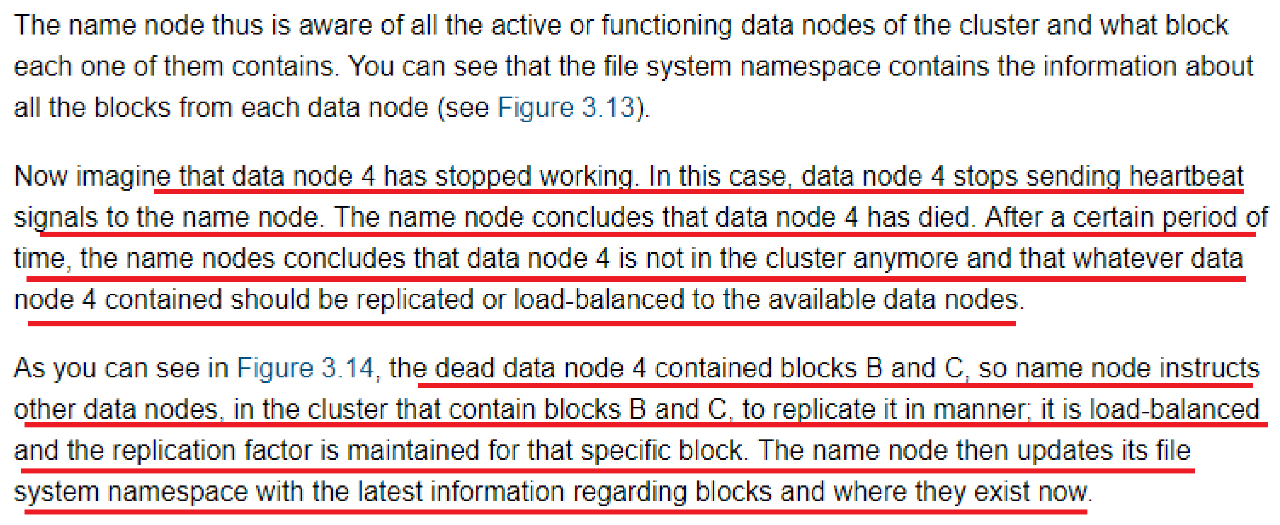
Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

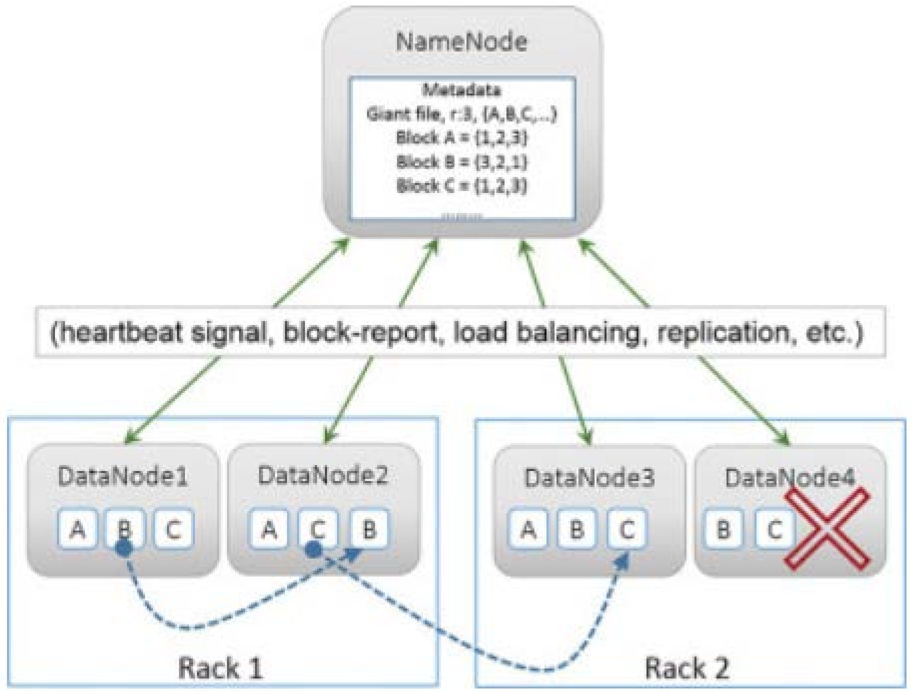
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Safemode

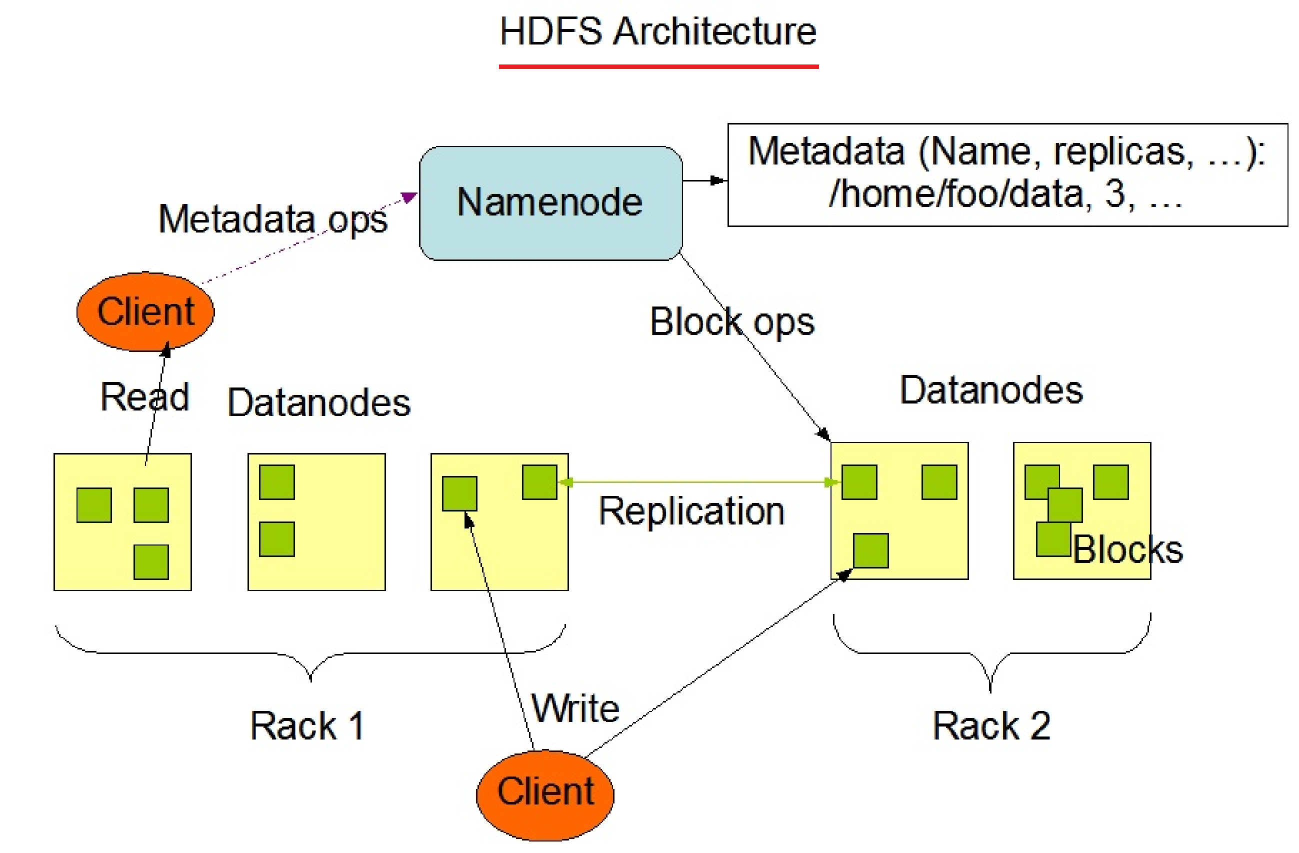
	 *Functions of NameNode:* • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata: ○ **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode. ○ **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

	 Hot standby is a redundant method in which one system runs simultaneously with an identical primary system. Upon failure of the primary system, the hot standby system immediately takes over, replacing the primary system. However, data is still mirrored in real time. Thus, both systems have identical data. Hot standby is also known as hot spare, especially at the component level, such as a hard drive in a disk array. https://www.techopedia.com/definition/1024/hot-standby

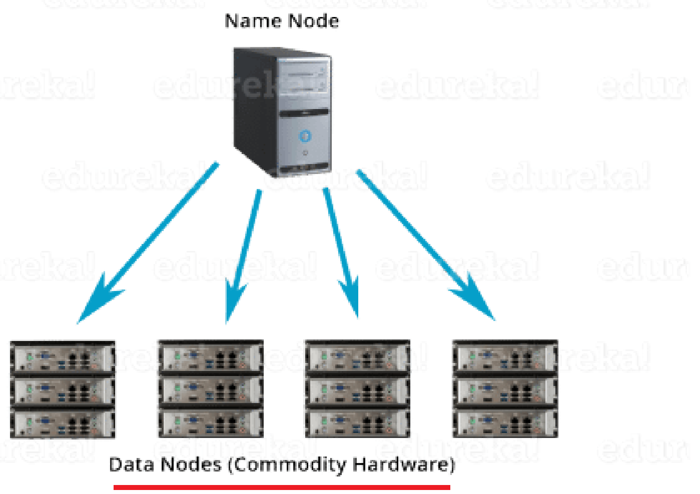
	Now let's examine how the name node handles a data node failure. In Figure 3.13, you can see four data nodes (two data nodes in each rack) in the cluster. These data nodes periodically send heartbeat signals (implying that a particular data node is active and functioning properly) and a block-report (containing a list of all blocks on that specific data node) to the name node.  NameNode Metadata Giant file, r:3, {A,B,C,...} Block A = {1,2,3} Block B = {4,3,1} Block C = {1,2,4} (heartbeat signal, block-report, load balancing, replication, etc.) DataNode1 A B C DataNode2 A C DataNode3 A B DataNode4 B C Rack 1 Rack 2 http://www.informit.com/articles/article.aspx?p=2460260&seqNum=2

	 The name node thus is aware of all the active or functioning data nodes of the cluster and what block each one of them contains. You can see that the file system namespace contains the information about all the blocks from each data node (see Figure 3.13). Now imagine that data node 4 has stopped working. In this case, data node 4 stops sending heartbeat signals to the name node. The name node concludes that data node 4 has died. After a certain period of time, the name nodes concludes that data node 4 is not in the cluster anymore and that whatever data node 4 contained should be replicated or load-balanced to the available data nodes. As you can see in Figure 3.14, the dead data node 4 contained blocks B and C, so name node instructs other data nodes, in the cluster that contain blocks B and C, to replicate it in manner; it is load-balanced and the replication factor is maintained for that specific block. The name node then updates its file system namespace with the latest information regarding blocks and where they exist now. http://www.informit.com/articles/article.aspx?p=2460260&seqNum=2

	 NameNode Metadata Giant file, r:3, {A,B,C,...} Block A = {1,2,3} Block B = {3,2,1} Block C = {1,2,3} (heartbeat signal, block-report, load balancing, replication, etc.) DataNode1 DataNode2 DataNode3 DataNode4 A B C A C B A B C B C Rack 1 Rack 2 http://www.informit.com/articles/article.aspx?p=2460260&seqNum=2
controlling the communication between the each of the pair switching system and a monitoring unit in accordance with the an operating state of the respective switching system;	The system, at least in internal testing and usage, utilized by the accused product practices controlling the communication between the each of the pair switching system (e.g., distributed or remote racks for Datanodes) and a monitoring unit (e.g., Namenode) in accordance with the operating state (e.g., active or hot-standby) of the respective switching system. The monitoring unit (i.e., Namenode) monitors status and health of the data nodes in different racks. Upon information and belief, the system comprises a controlling unit or administrative unit which configures and manage Namenode services and control communication between the Namenode and the Data nodes.

	 HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, …): /home/foo/data, 3, … Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

HDFS Architecture:



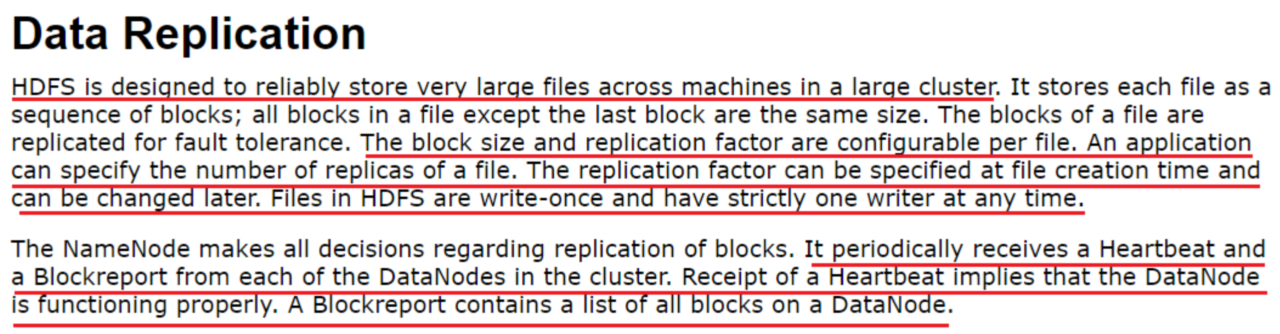
Apache HDFS or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

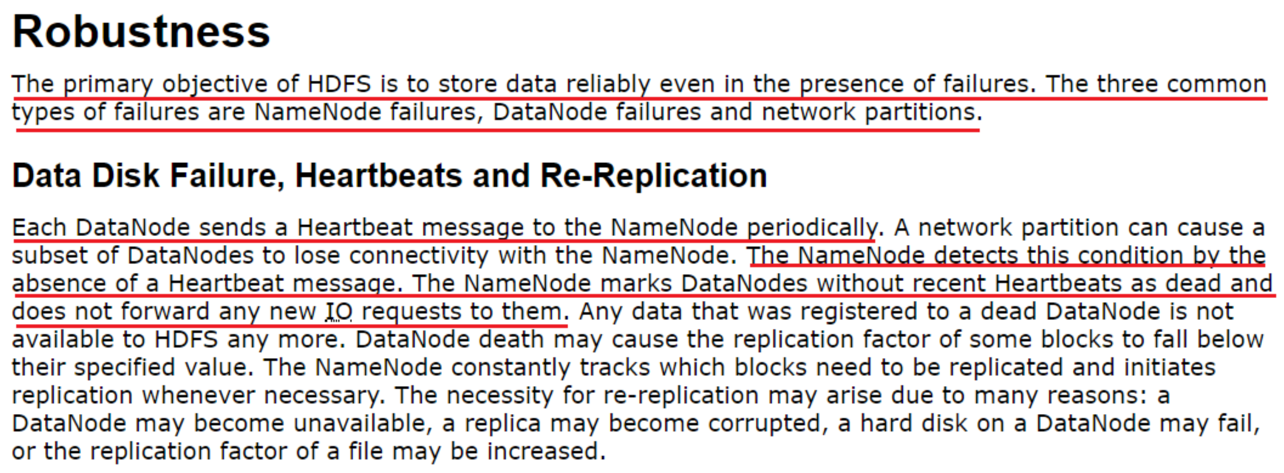


Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html



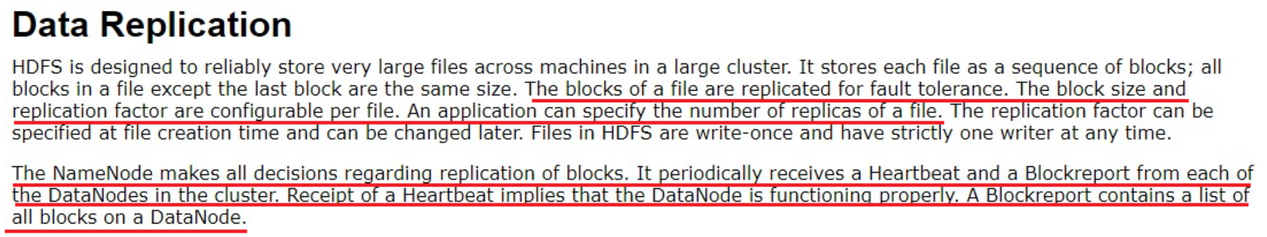
Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

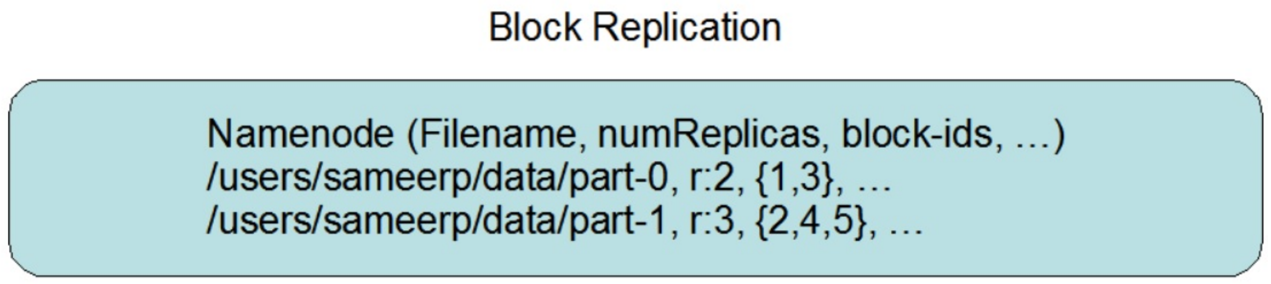
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Disk+Failure%2C+Heartbeats+and+Re-Replication



Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

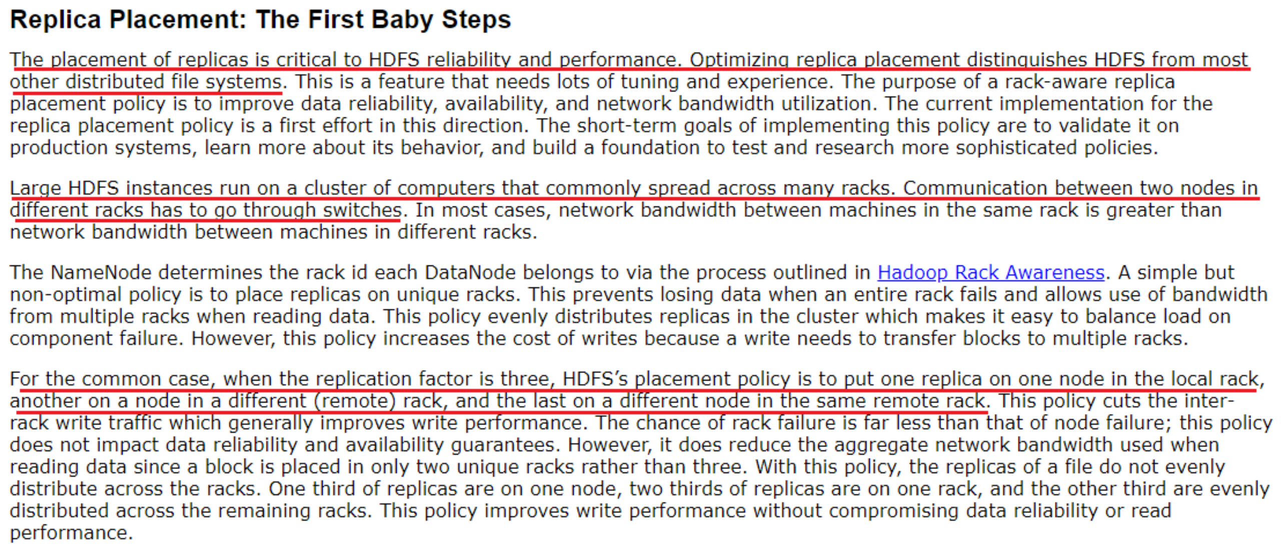
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.



Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



Replica Placement: The First Baby Steps

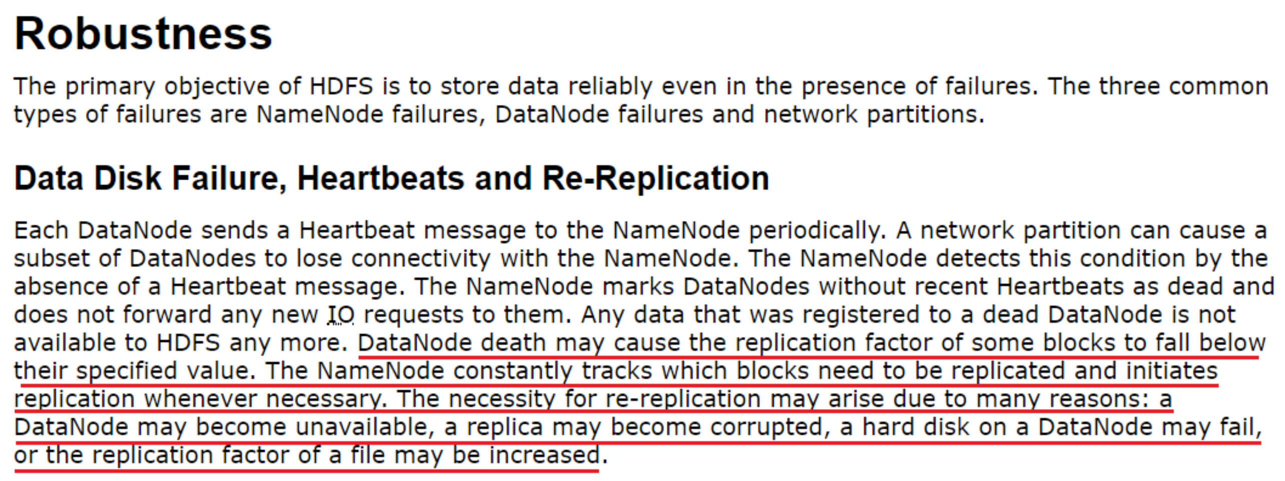
The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



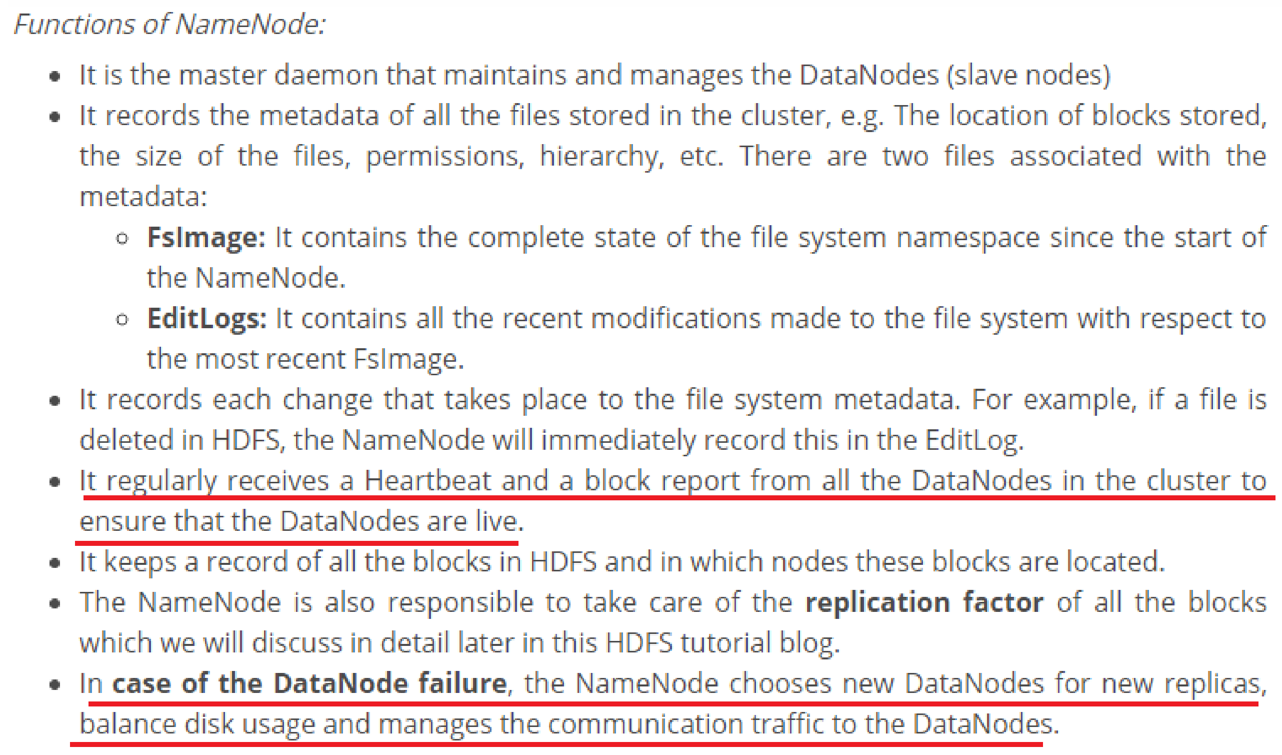
Robustness

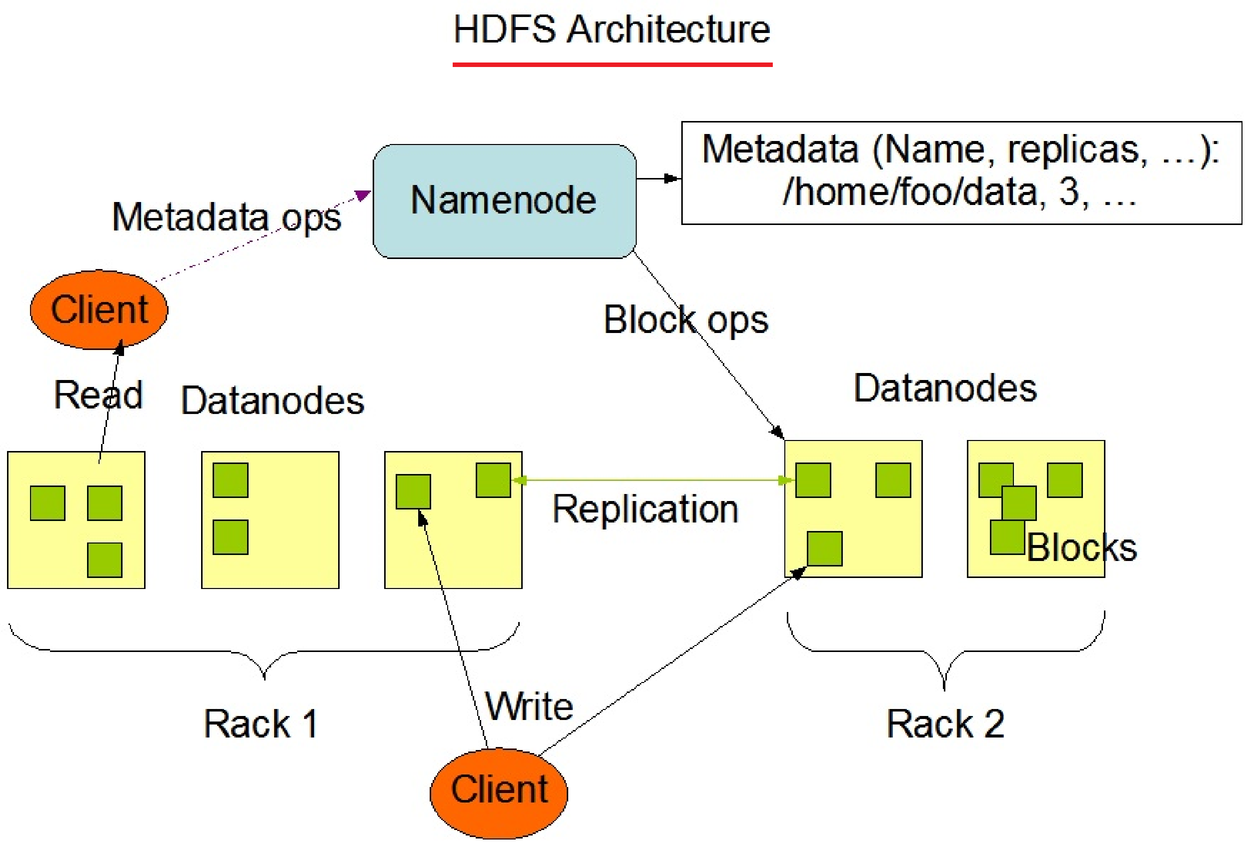
The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

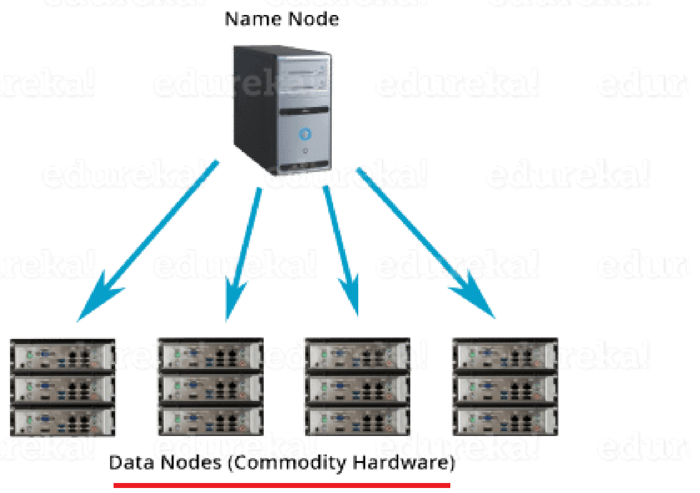
Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Safemode

	 *Functions of NameNode:* • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata: ○ **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode. ○ **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/
when a loss of the communication to the switching system in the active operating state occurs:	The system, at least in internal testing and usage, utilized by the accused product practices determining a loss of the communication to the switching system in the active operating state (e.g., a data node failure in a rack). The monitoring unit (i.e., Namenode) monitors status and health of the data nodes in different racks. Each data node sends a periodic heartbeat message to the Namenode. The Namenode marks a data node as dead or lost when doesn't receive a heartbeat message from the node.

	 HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, ...): /home/foo/data, 3, ... Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

HDFS Architecture:



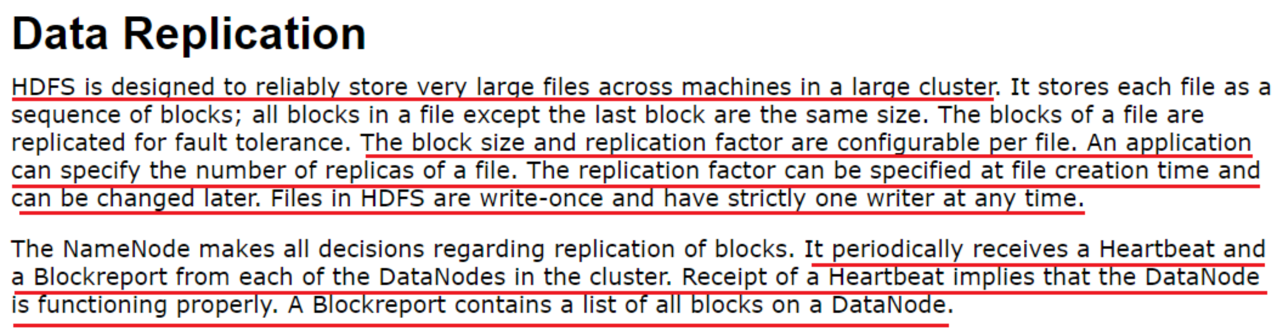
Apache HDFS or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

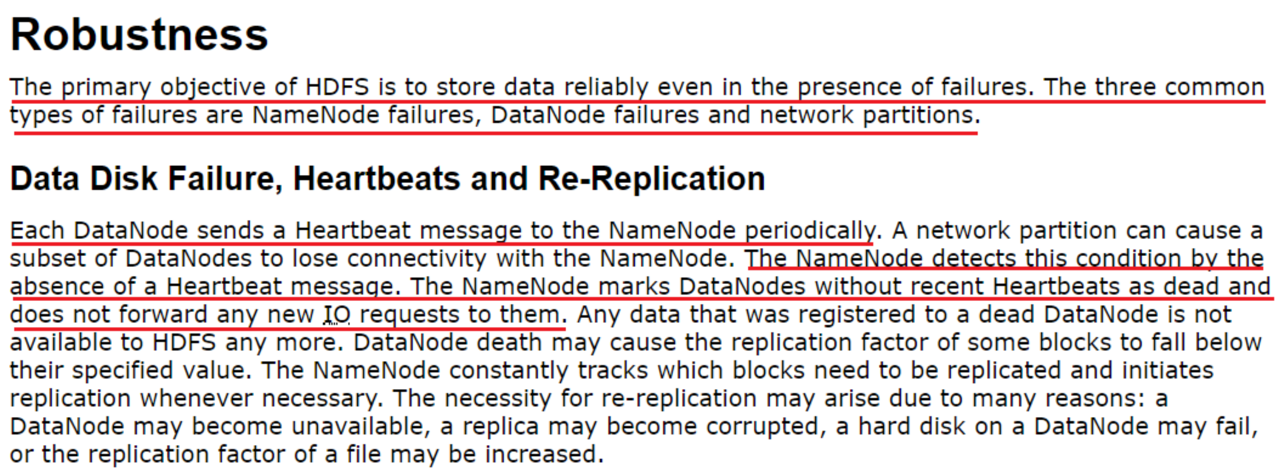


Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html



Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

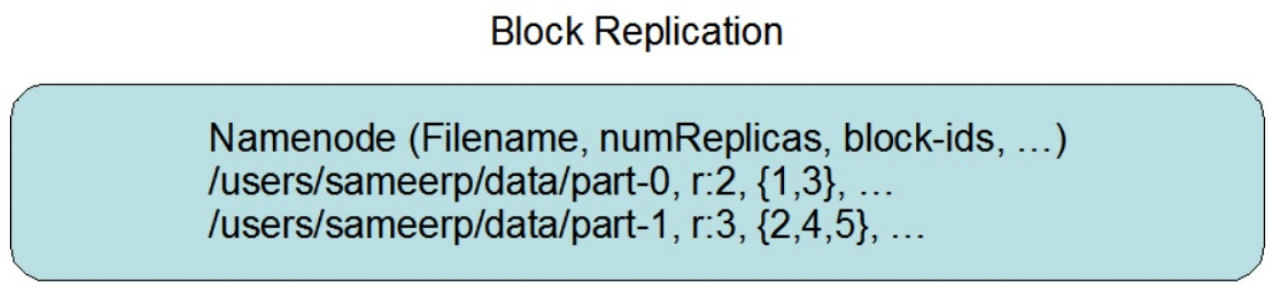
Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Disk+Failure%2C+Heartbeats+and+Re-Replication

Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.



Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication

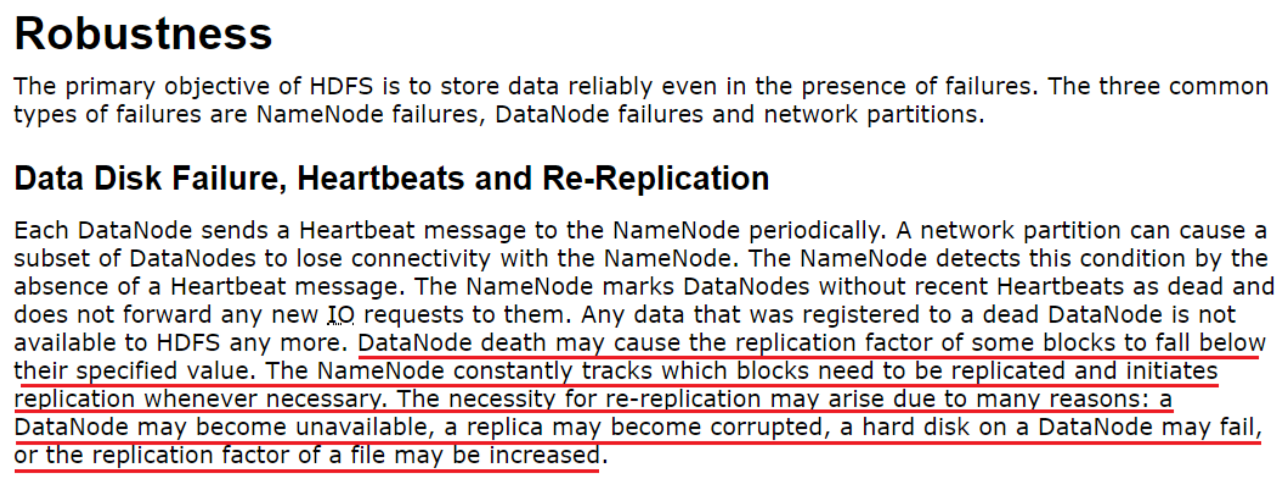
Replica Placement: The First Baby Steps

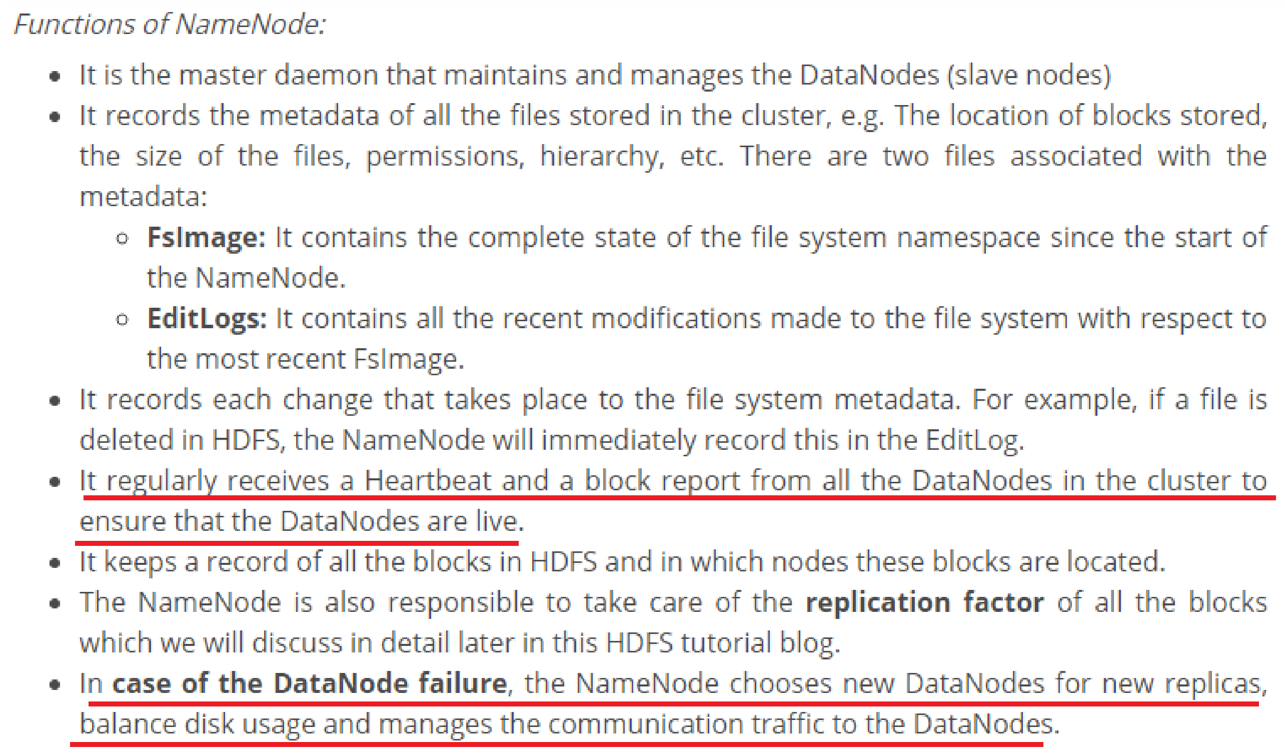
The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

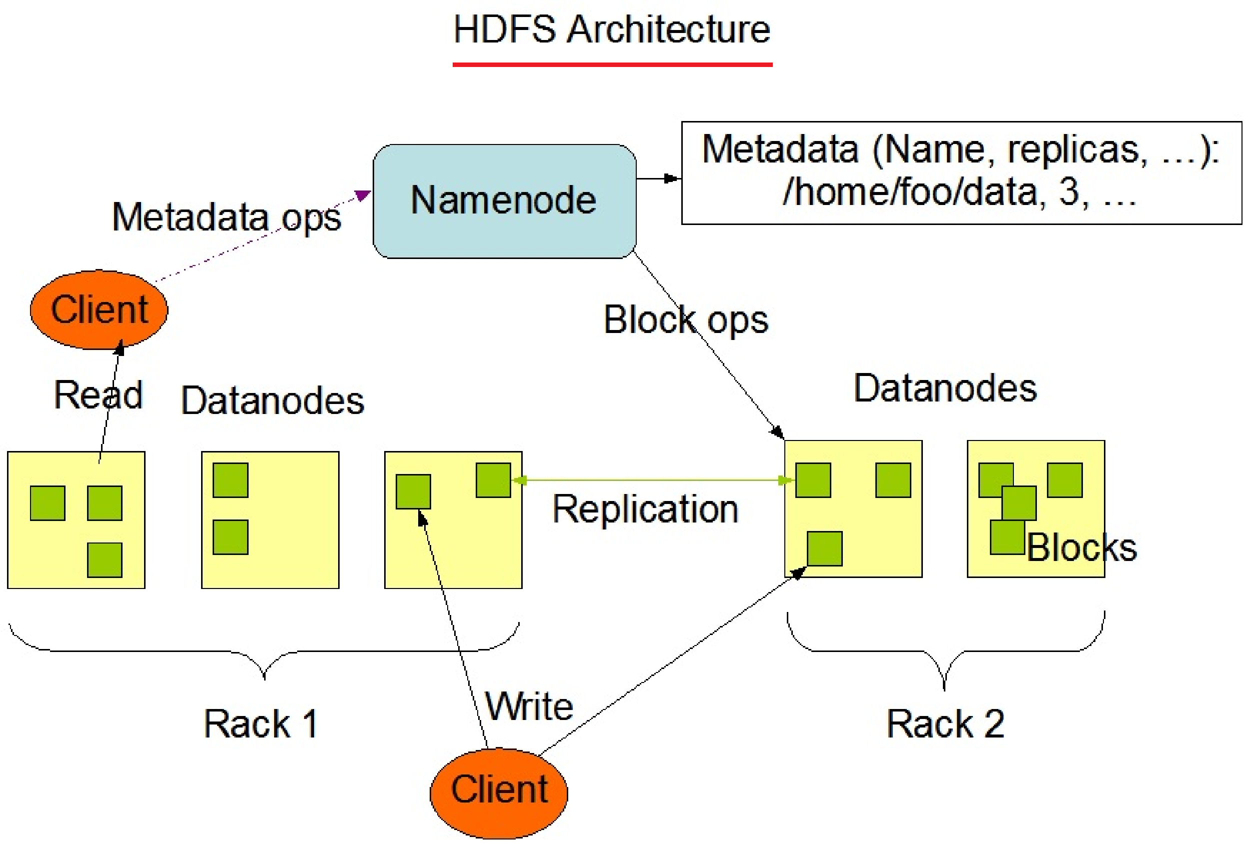
The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

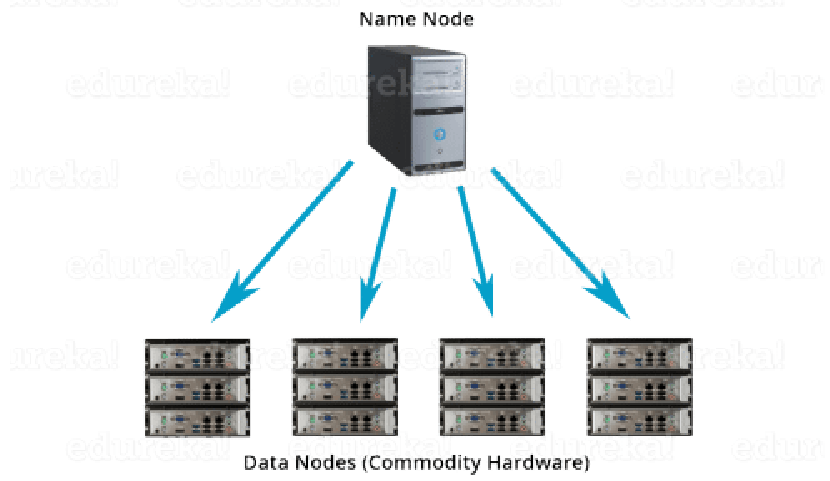
	https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication  **Robustness** The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions. **Data Disk Failure, Heartbeats and Re-Replication** Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Safemode

	 *Functions of NameNode:* • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata: ◦ **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode. ◦ **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/
activating, by the monitoring unit, the switching system in the hot-standby operating state to be in the active operating state, and deactivating, by the monitoring unit, the switching system with the communication loss to be in the hot-standby operating	The system, at least in internal testing and usage, utilized by the accused product practices activating, by the monitoring unit (e.g., Namenode server), the switching system (e.g., data nodes in different racks) in the hot-standby operating state to be in the active operating state, and deactivating, by the monitoring unit, the switching system with the communication loss to be in the hot-standby operating state, wherein when in the hot-standby operating state, the respective switching system is not active in terms of switching functions; and further features: periodically sending an IP lease request to the monitoring unit by a packet-based interface of the switching system in the hot-standby operating state, the packet-based interface is in an inactive state.

state, wherein when in the hot-standby operating state, the respective switching system is not active in terms of switching functions; and further features: periodically sending an IP lease request to the monitoring unit by a packet-based interface of the switching system in the hot-standby operating state, the packet-based interface is in an inactive state.	As shown below, the system utilized by the accused product comprises a Namenode server which monitors status and health of data nodes at different racks. Each data node sends periodic heartbeat message and block report to the Namenode server. When the Namenode server doesn't receive heartbeat message from the data node at a local rack, it determines that the data node in the rack is lost or not available and changes the status of the data node. The Namenode manages communication traffic and disk usages with the data nodes at remote rack. Upon information and belief, The Namenode (i.e., monitoring unit) switches states of rack pair, the data node pair at local rack is considered as lost or dead and the data nodes at remote rack are used primarily to manage traffic. The data node at the remote rack periodically pings the Namenode for network resources to communicate with a client device. The data node sends an IP lease request to the monitoring unit (e.g., Namenode).

	 HDFS Architecture Metadata ops Namenode Metadata (Name, replicas, …): /home/foo/data, 3, … Client Block ops Read Datanodes Datanodes Replication Blocks Rack 1 Write Rack 2 Client https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

HDFS Architecture:



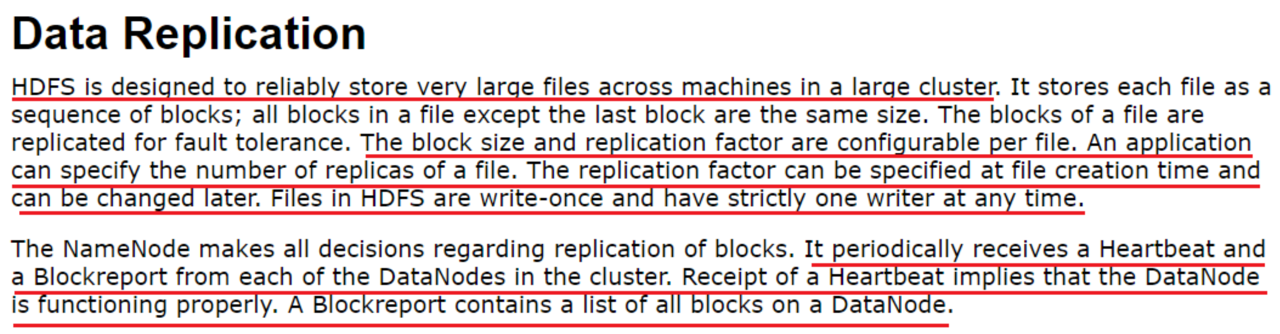
Apache HDFS or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

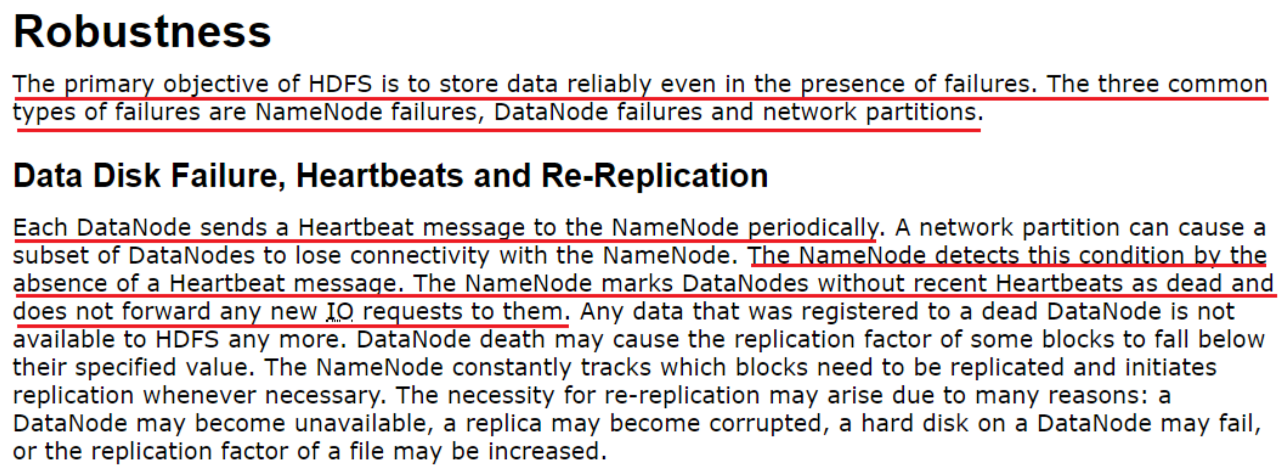


Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html



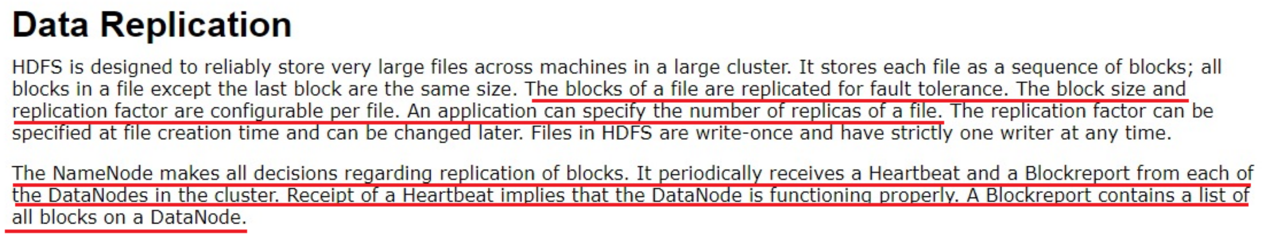
Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

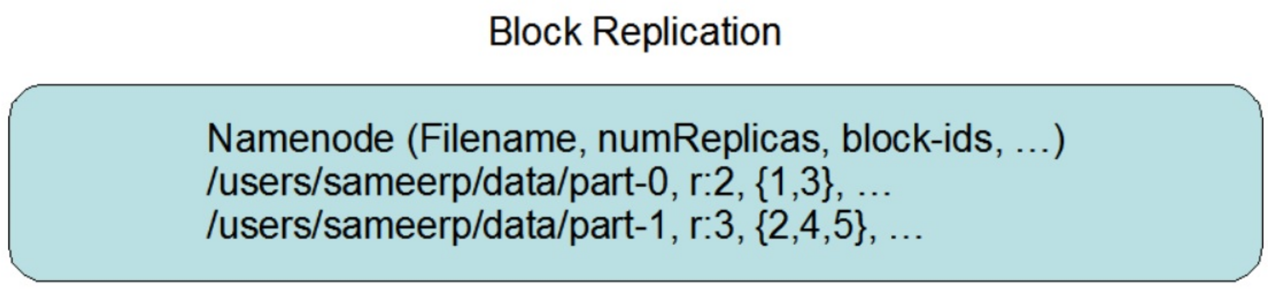
https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Disk+Failure%2C+Heartbeats+and+Re-Replication



Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

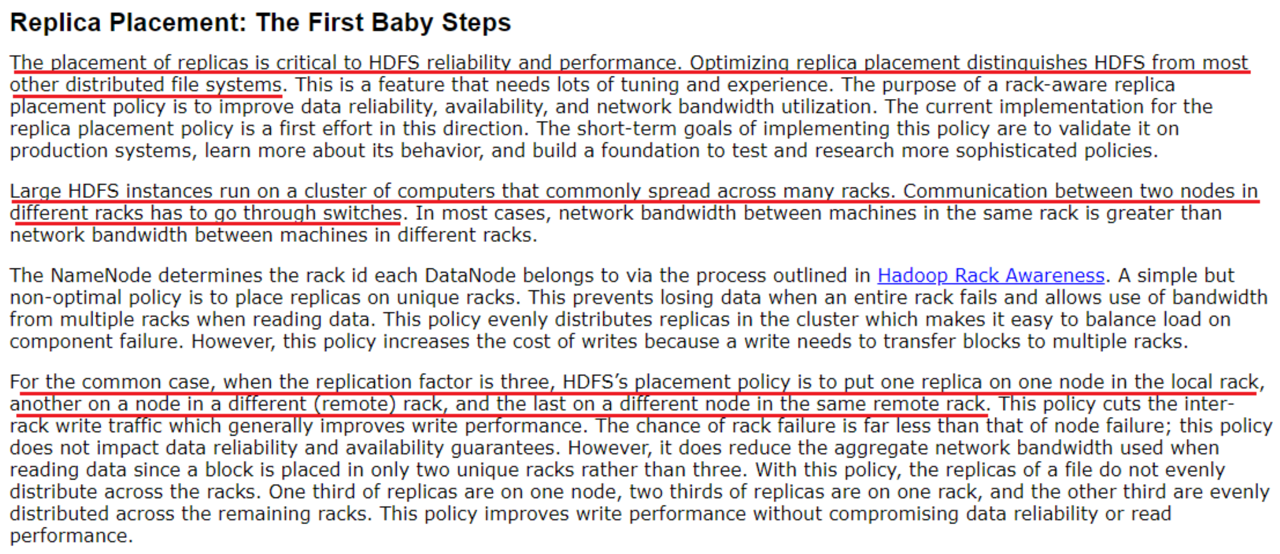
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.



Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication



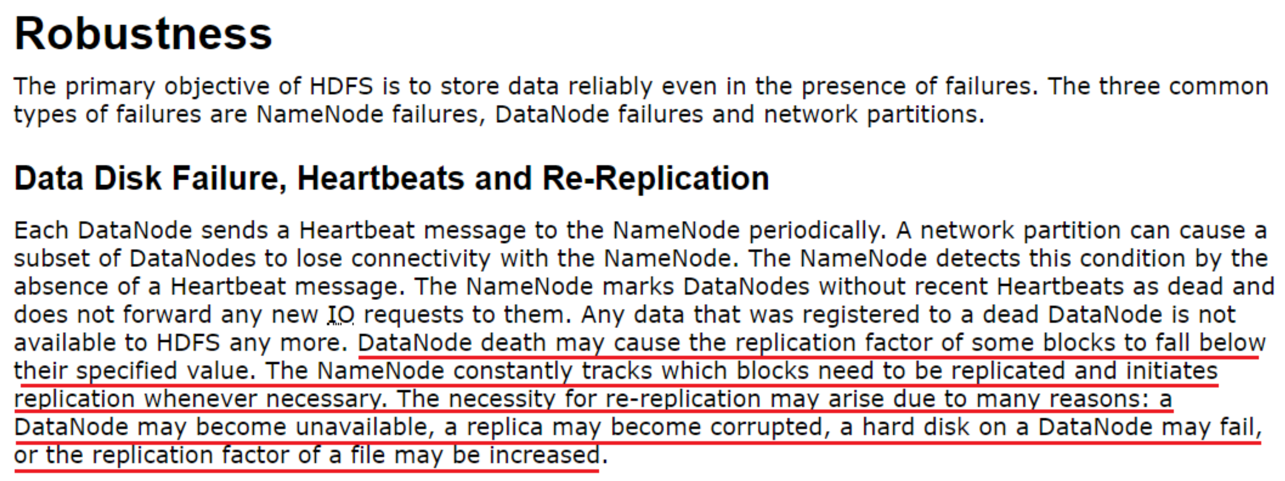
Replica Placement: The First Baby Steps

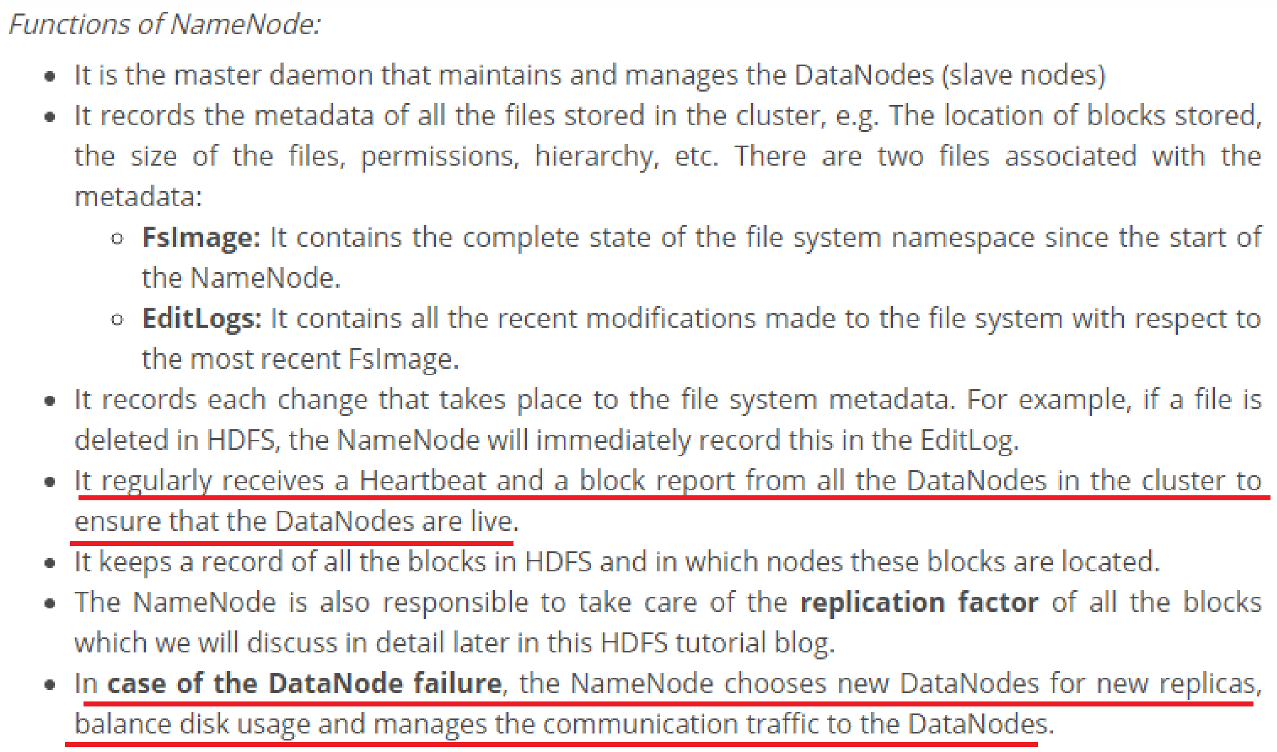
The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

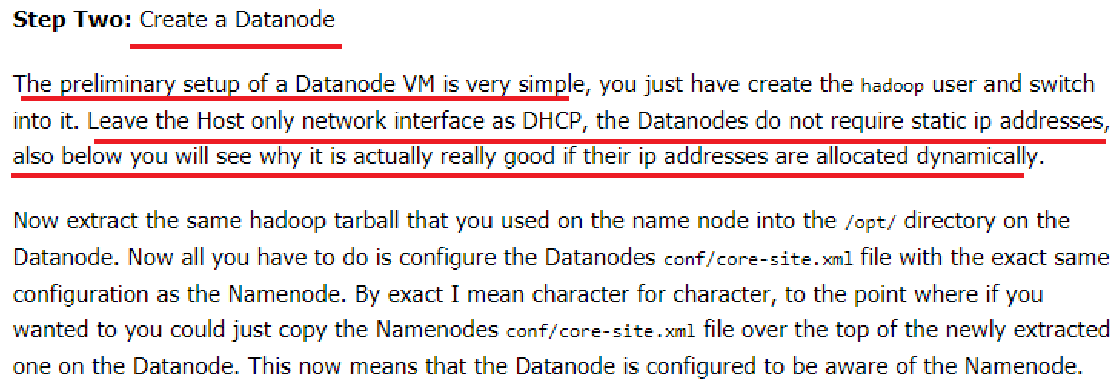
Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS's placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

	https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication  **Robustness** The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions. **Data Disk Failure, Heartbeats and Re-Replication** Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Safemode

	 *Functions of NameNode:* • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata: ○ **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode. ○ **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

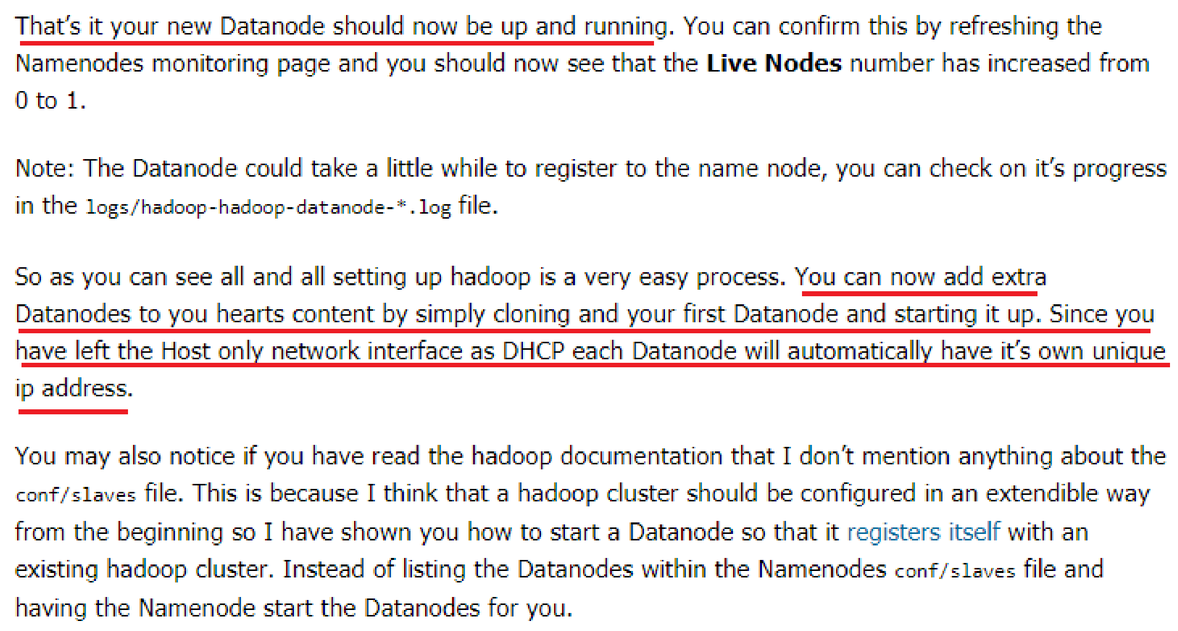


Step Two: Create a Datanode

The preliminary setup of a Datanode VM is very simple, you just have create the `hadoop` user and switch into it. Leave the Host only network interface as DHCP, the Datanodes do not require static ip addresses, also below you will see why it is actually really good if their ip addresses are allocated dynamically.

Now extract the same hadoop tarball that you used on the name node into the `/opt/` directory on the Datanode. Now all you have to do is configure the Datanodes `conf/core-site.xml` file with the exact same configuration as the Namenode. By exact I mean character for character, to the point where if you wanted to you could just copy the Namenodes `conf/core-site.xml` file over the top of the newly extracted one on the Datanode. This now means that the Datanode is configured to be aware of the Namenode.

https://pier0w.wordpress.com/2012/02/23/hadoop-hdfs/

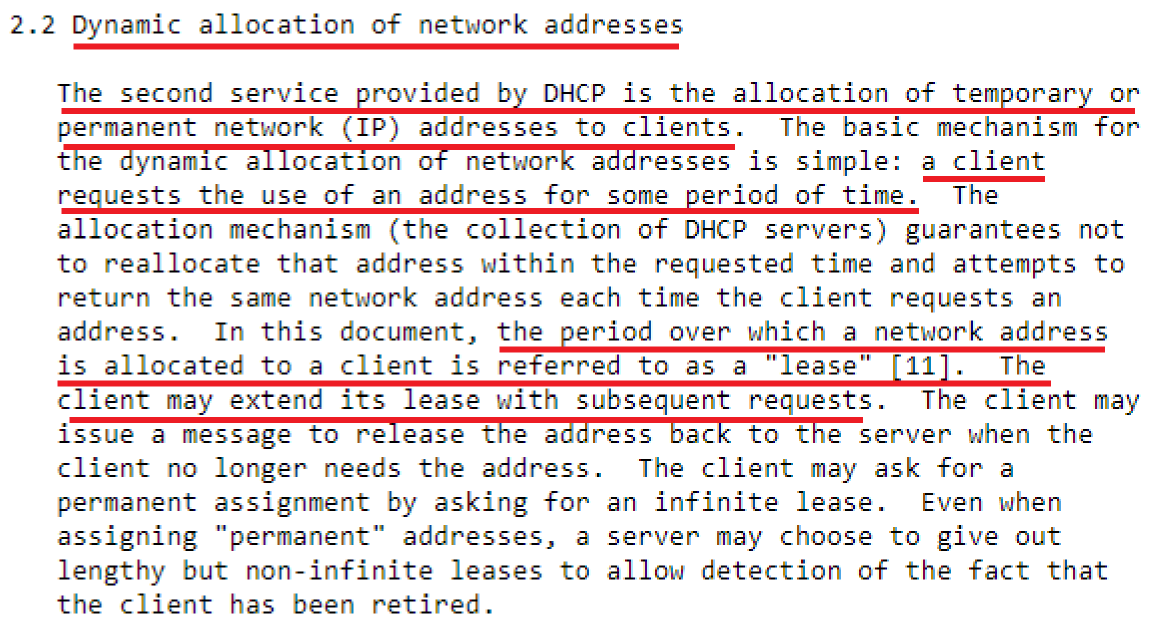


That's it your new Datanode should now be up and running. You can confirm this by refreshing the Namenodes monitoring page and you should now see that the **Live Nodes** number has increased from 0 to 1.

Note: The Datanode could take a little while to register to the name node, you can check on it's progress in the `logs/hadoop-hadoop-datanode-*.log` file.

So as you can see all and all setting up hadoop is a very easy process. You can now add extra Datanodes to you hearts content by simply cloning and your first Datanode and starting it up. Since you have left the Host only network interface as DHCP each Datanode will automatically have it's own unique ip address.

You may also notice if you have read the hadoop documentation that I don't mention anything about the `conf/slaves` file. This is because I think that a hadoop cluster should be configured in an extendible way from the beginning so I have shown you how to start a Datanode so that it registers itself with an existing hadoop cluster. Instead of listing the Datanodes within the Namenodes `conf/slaves` file and having the Namenode start the Datanodes for you.

	https://pier0w.wordpress.com/2012/02/23/hadoop-hdfs/  2.2 Dynamic allocation of network addresses The second service provided by DHCP is the allocation of temporary or permanent network (IP) addresses to clients. The basic mechanism for the dynamic allocation of network addresses is simple: a client requests the use of an address for some period of time. The allocation mechanism (the collection of DHCP servers) guarantees not to reallocate that address within the requested time and attempts to return the same network address each time the client requests an address. In this document, the period over which a network address is allocated to a client is referred to as a "lease" [11]. The client may extend its lease with subsequent requests. The client may issue a message to release the address back to the server when the client no longer needs the address. The client may ask for a permanent assignment by asking for an infinite lease. Even when assigning "permanent" addresses, a server may choose to give out lengthy but non-infinite leases to allow detection of the fact that the client has been retired. https://www.ietf.org/rfc/rfc2131.txt